MICRO™

The Magazine of the APPLE, KIM, PET

and Other 6502 Systems

PRETTY PET PICTURES

NO 10 March 1979 $1.50

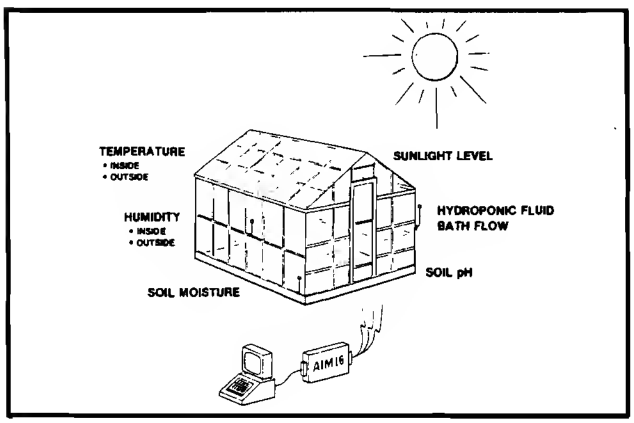
DAM YOUR PET
DAM YOUR TRS-80
DAM YOUR KIM
DAM YOUR . . .

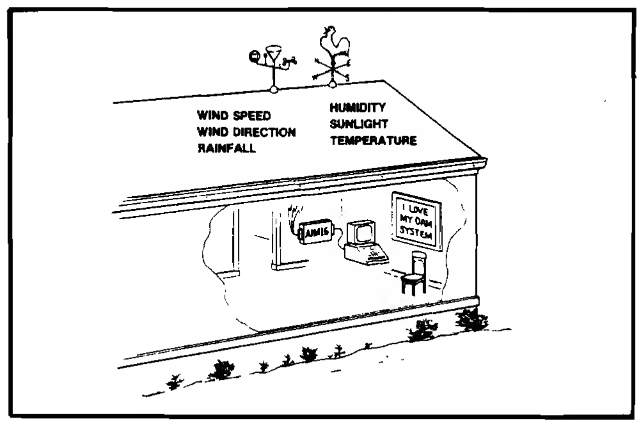
MEASURE - RECORD - CONTROL

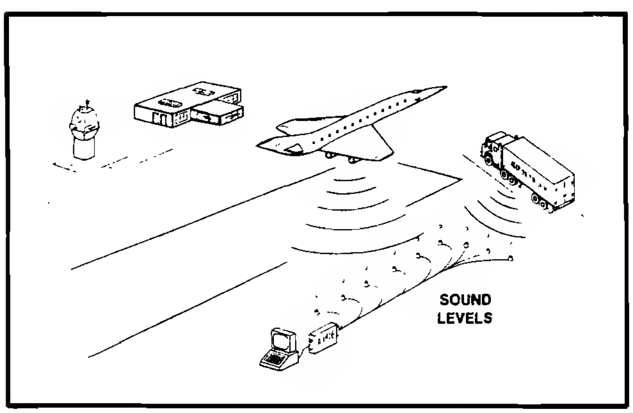
- TEMPERATURE
- DIRECTION
- PRESSURE
- LIGHT LEVELS
- db
- POLLUTION CONTROLS
- DARKROOMS
- HUMIDITY
- LIGHT
- ENERGY CONSERVATION EQUIPMENT
- GREENHOUSES
- SPEED
- WEATHER STATIONS
- NOISE POLLUTION
- pH
- EARTHQUAKE TREMORS
- VELOCITY
- ACCELERATION

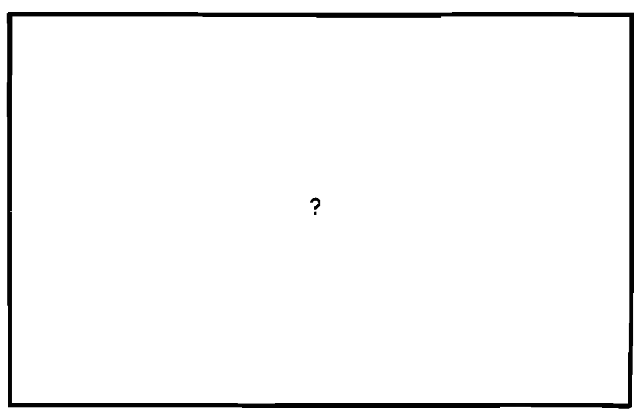
DATA
ACQUISITION by CmC
MODULES

NOW YOUR COMPUTER CAN LISTEN TO THE REAL WORLD. YOU GET 16 8 BIT ANALOG INPUTS WITH OUR AIM16.









CONNECTICUT microCOMPUTER
150 POCONO ROAD · BROOKFIELD, CONNECTICUT 06804
(203) 775-9659

DAM SYSTEMS

MICRO™

MARCH 1979
ISSUE NUMBER TEN

TABLE OF CONTENTS

MICRO™

STAFF

Editor/Publisher
Robert M. Tripp

Business Manager
Donna M. Tripp

Administrative Assistant
Susan K. Lacombe

Circulation Manager
Maggie Fisher

Distribution
Eileen M. Enos
Carol A. Stark

Micro-Systems Lab
James R. Witt

Gofer
Fred Davis

MICRO^tm is published monthly by:

The COMPUTERIST, Inc.
P.O. Box 3
So. Chelmsford, MA 01824

Controlled Circulation postage paid at:

Chelmsford, MA 01824

Publication Number: COTR 395770

Subscription in US: $12.00/12 Issues

ADVERTISER'S INDEX

Please address all correspondence, subscriptions and address changes to:

MICRO, P.O. Box 3, So. Chelmsford, MA 01824

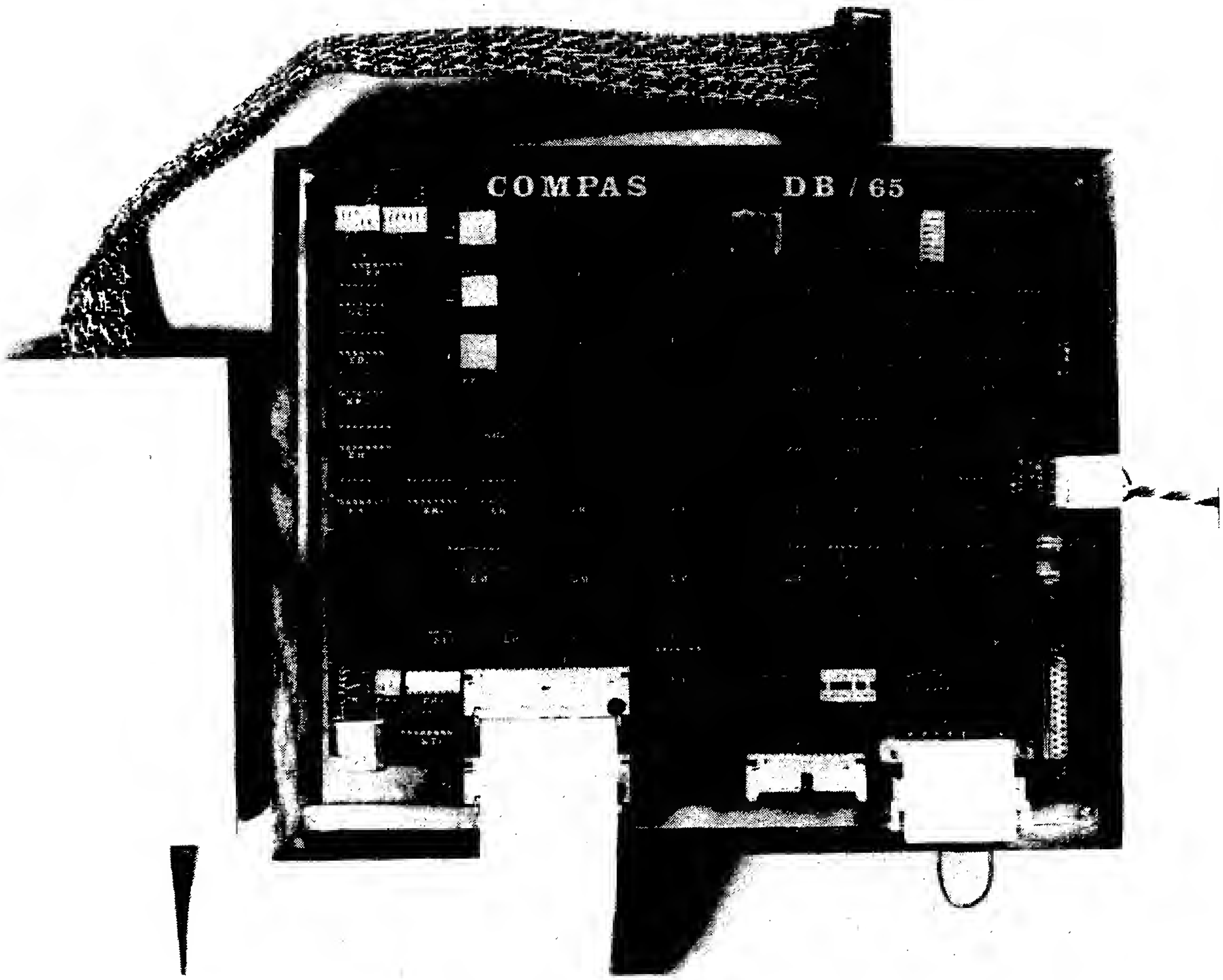
compas microsystems

P.O. Box 687 224 S.E. 16th Street Ames, Iowa 50010

515/232-8187

DB/65

A complete hardware/software debug system for the Rockwell, Synertek, MOS/Technology 6500 microprocessor family.



Features

* Standard in-circuit emulator
* Hardware breakpoints
* Single step mode
* Eight software breakpoints
* Real-time software breakpoints
* RS 232C or current loop terminals
* Symbolic disassembly of user program
* Serial/parallel load capability
* Program trace of instructions and registers
* Prom resident debug monitor
* Software history of instruction addresses
* 2K ram standard with sockets for additional 6K if required
* Scope sync output
* User NMI and IRQ vectors supported
* Write protect
* User program may reside in high memory

SINGLE QUANTITY PRICE = $1450

IN THIS ISSUE ...

Marvin L. De Jong, who in previous issues of MICRO presented a fine series on interfacing the 6502 to peripherals, now tackles the AIM 65 and provides "A Simple 24 Hour Clock for the AIM 65" (page 5). The program listings are given in two formats: first, the format as produced by the AIM Disassembler, and second, the more "normal" commented listing which was produced by the MICRO Staff. We expect to see a lot more on the AIM 65 is the near future.

Alan G. Hill provides an "Apple II Trace List Utility" (page 9) which should certainly make it much easier to debug BASIC programs on an Apple. This features two modes of operation - a "real-time" mode and a mode which "remembers" the last 100 line executed for post-execution examination.

The ever popular MICRO Software Catalog (page 15) continues to provide a source of information on what software is available for the 6502.

John T. Kosinski and Richard F. Suitor put to rest forever (hopefully) the discussion of the 6522 I/O chip and the Apple II in their article on "6522 Chip Setup Time" (page 17).

John R. Sherburne shows how you can combine the serious and the enjoyment aspects of your system with his article on "High-Resolution Plotting for the PET" (page 19). The cover photo for this issue of MICRO came from his work.

Jim Zuber turns the tables by "Using Tiny BASIC to Debug Machine Language Programs" (page 25). While we have had a number of articles which used machine language programs to support BASIC, this is the first I can remember which used BASIC to support machine language programming.

Robert M. Tripp continues to "ASK the Doctor" with "An ASK EPROM Programmer" (page 31). This program, in addition to being useful in its own right, shows how to write code that can run on an AIM, SYM, and KIM.

Harvey B. Herman wrote "Thanks for the Memories A Pet Machine Language Memory Test" (page 37) which provides a very efficient memory test for the PET.

Robert E. Jones presents the first OSI-based article we have had since I twisted the arm of a friend back in the earliest days of MICRO. "The OSI Flasher: BASIC-Machine Code Interfacing" (page 41) presents an example of how to get into a machine language program from BASIC and back again.

Jim Green provides some "6502 Graphics Routines" (page 43) which will help anyone who is trying to do low resolution graphics on a KIM-based system with video capability.

William R. Dial, recently returned from a vacation in Hawaii, continues to keep us informed on what is being published about the 6502 in his "6502 Bibliography - Part IX" (page 47).

MICRO INTERRUPTS

MICROBES

In "Life for the KIM-1 ..."" by T. E. Bridge in MICRO 9:39, you need to insert $1C00 in the IRQ vector - as any good Kimmy knows, not $1000 as was listed.

The real author of "6502 Interfacing ..." in MICRO 9:11 was not Martin L. De Jong as listed, but rather Marvin L. De Jong, but you probably already guessed that.

ANNOUNCEMENTS

A number of readers have written or called for the address of CALL APPLE which we have mentioned in some previous issues. It is:

CALL APPLE
6708 39th Avenue S.W.
Seattle, WA 98136
206/932-6588

Another Apple group has been formed:

Apple Group - NJ
Amateur Computer Group of NJ
c/o Steve Toth
1411 Greenwood Drive
Piscataway, NJ 08854
201/96807498

Connecticut microComputer is offering two free application notes to MICRO readers.
1. Software Delay for Slow Carriage Return Printers Using the CmC ADA 1200 C
2. Output Formatting Routines for the PET
You will find the address for CmC in their ad inside the front cover of this issue.

Classified Advertisements

Starting with the next issue of MICRO, we will print ads from individuals, stores, companies, etc. in a classified ad format. The rules are:

1. Ad must pertain to 6502 material.
2. Ad must not exceed 250 characters.
3. $10.00 fee must accompany ad.
4. Only one ad per issue per person/store.

The purpose of this advertising service is to permit anyone who has things to sell to the 6502 community to be able to advertise at a very low rate while getting their message to over 6000 interested MICRO readers. We reserve the right to reject any material which we feel is not in the best interest of MICRO or its readers.

Why Do Stores Get MICRO First?

A number of subscribers have complained that the local computer store gets MICRO several weeks before they receive theirs through the mail. The problem is simply the slow delivery of 2nd Class mail. All copies of MICRO, both dealer which are shipped UPS and subscribers which are mailed, leave us within a two or three day span. The February issue of MICRO was mailed on the 26th of January. It took exactly two weeks for a reader in the next town to receive his copy. First Class postage would cost $8.40 more per year, based on the current size and weight of MICRO.

A SIMPLE 24 HOUR CLOCK FOR THE AIM 65

Marvin L. De Jong
Department of Math-Physics
The School of the Ozarks
Point Lookout, MO 65726

The program whose listings are given in the AIM 65 disassembly format is a 24 hour clock that displays the time in hours, minutes, and seconds on the six right-most digits of the 20 character AIM 65 display. AIM 65 owners can load the program directly from the listings using the mini-assembler in the AIM 65 monitor. The program listings were taken directly from the thermal printer on the AIM 65.

The principal reason for writing the program was to experiment with the interval timers on the 6522 VIA. One advantage of the so-called T1 timer on the 6522 is that it can produce equally spaced interrupts, independent of the time necessary to complete an instruction and the time necessary to process the interrupt. SYM-1 owners may also use the program with only minor modifications, since the addresses of the various registers and counters in the 6522 chips are the same for these two computers. SYM-1 owners will have to change the display routines, however.

A brief description of the program follows. The first five instructions set up the interrupt vectors for the AIM 65. The next eight instructions set up the 6522 VIA for the T1 timer in the free running mode, enable the T1 interrupt, and set the time interval to \$C34E = $49{,}998_{10}$ clock cycles. This number, plus the two clock cycles necessary to restart the timer, represent 50,000 clock cycles or 0.05 seconds. Thus, the time between interrupts is exactly 50,000 clock cycles. Twenty interrupts give an interval of 10^6 clock cycles, or one second with a one MHz clock frequency. Location \$0000 serves as register for the count-to-twenty interrupts process. It starts at \$EC and advances to \$00 before the seconds location is incremented.

The interrupt routine from \$0300 to \$033C is very similar to the clock program by Charles Parsons in THE FIRST BOOK OF KIM. The only difference is that the timers do not need to be restarted in the interrupt routine. Only the interrupt flag needs to be cleared before returning from interrupt. This is accomplished by the LDA A004 instruction at \$0337.

The program from \$0226 to \$0254 is the display routine from the AIM 65. First the seconds, minutes, and hours located in \$0001, \$0002, and \$0003 respectively, are relocated, then converted to ASCII, and finally output to the display by the JSR EF7B. Many kinds of hex to ASCII routines are possible here. I simply rotated nibble after nibble into the low order nibble of location \$0004 and added \$30 to convert to ASCII.

AIM 65 owners may be interested in the output routine. Of all the subroutines mentioned in the "User's Guide" the one I used is not mentioned directly. Basically it takes an ASCII character in the accumulator and outputs it to the display digit between \$00 and \$13 (20 character display) identified by the contents of the X register. It also requires a one in bit seven of the accumulator. Otherwise you get the cursor. So I did a ORA \$80 with the ASCII character in the accumulator before jumping to the subroutine at \$EF7B.

I checked the clock up against WWV and found it was off by about 0.024%, which is substantial if you wish to keep time over the long term. I decreased the \$4E byte location \$0216 to \$42 and now it appears to be off by only 0.00063%. Of course, these timing errors, though small, tend to accumulate giving an error of about 0.5 seconds in 24 hours.

To start the timer, load the hours, minutes, and seconds locations with the time at which you intend to start, wait for this time, then start the program. Of course, there are much more meaningful applications to this program than simply displaying the time. One could record the time at which transistions on the I/O pins occur for example. Have fun.

```
0200 78 SEI
0201 A9 LDA #00
0203 8D STA A404
0206 A9 LDA #03
0208 8D STA A405
020B A9 LDA #C0
020D 8D STA A00E
0210 A9 LDA #40
0212 8D STA A00B
0215 A9 LDA #4E
0217 8D STA A006
021A A9 LDA #C3
021C 8D STA A005
021F A9 LDA #EC
0221 85 STA 00
0223 58 CLI
0224 00 BRK
0225 EA NOP
0226 A5 LDA 01
0228 85 STA 04
022A A5 LDA 02
022C 85 STA 05
022E A5 LDA 03
0230 85 STA 06
0232 A2 LDX #13
0234 8A TXA
0235 48 PHA
0236 A0 LDY #04
0238 A5 LDA 04
023A 29 AND #0F
023C 18 CLC
023D 69 ADC #30
023F 09 ORA #80
0241 20 JSR EF7B
0244 46 LSR 06
0246 66 ROR 05
0248 66 ROR 04
024A 88 DEY
024B D0 BNE 0244
024D 68 PLA
024E AA TAX
024F CA DEX
0250 E0 CPX #0E
0252 B0 BCS 0234
0254 4C JMP 0226
```

```
0300 48 PHA
0301 E6 INC 00
0303 D0 BNE 0337
0305 F8 SED
0306 18 CLC
0307 A5 LDA 01
0309 69 ADC #01
030B 85 STA 01
030D C9 CMP #60
030F 90 BCC 0333
0311 A9 LDA #00
0313 85 STA 01
0315 18 CLC
0316 A5 LDA 02
0318 69 ADC #01
031A 85 STA 02
031C C9 CMP #60
031E 90 BCC 0333
0320 A9 LDA #00
0322 85 STA 02
0324 18 CLC
0325 A5 LDA 03
0327 69 ADC #01
0329 85 STA 03
032B C9 CMP #24
032D 90 BCC 0333
032F A9 LDA #00
0331 85 STA 03
0333 A9 LDA #EC
0335 85 STA 00
0337 AD LDA A004
033A D8 CLD
033B 68 PLA
033C 40 RTI
```

24 HOUR AIM CLOCK

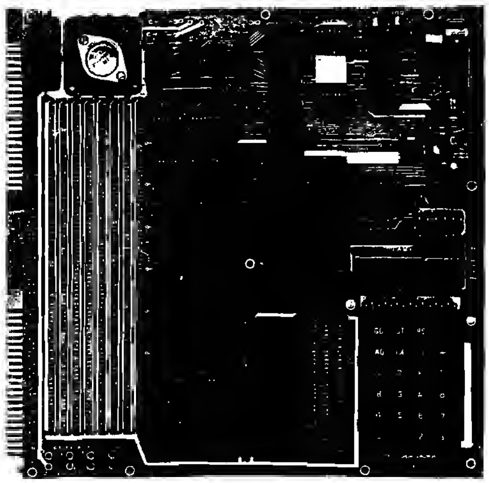
BY MARVIN L. DE JONG
FEBRUARY 1979

```
0200                         ORG    $0200

0200 78            START     SEI           SET INTERRUPT DISABLE
0201 A9 00                   LDAIM $00     SETUP INTERRUPT VECTORS
0203 8D 04 A4                STA   $A404   FOR 6522
0206 A9 03                   LDAIM $03     POINT TO ADDRESS 0300
0208 8D 05 A4                STA   $A405
020B A9 C0                   LDAIM $C0     SETUP VIA 6522 FOR TIMER 1
020D 8D 0E A0                STA   $A00E   IN FREE RUNNING MODE
0210 A9 4C                   LDAIM $40
0212 8D 0B A0                STA   $A00B
0215 A9 4E                   LDAIM $4E     SET LOW BYTE OF TIMER
0217 8D 06 A0                STA   $A006
021A A9 C3                   LDAIM $C3     SET HIGH BYTE OF TIMER
021C 8D 05 A0                STA   $A005
021F A9 EC                   LDAIM $EC.    SET 20 INTERRUPT COUNTER
0221 85 00                   STA   $0000   IN LOCATION 0000
0223 58                      CLI           ENABLE INTERRUPTS
0224 00                      BRK           RETURN TO MONITOR
0225 EA                      NOP

0226 A5 01         DISPLY    LDA   $0001   MOVE DIGITS TO BE DISPLAYED
0228 85 04                   STA   $0004   FOR SAFE KEEPING
022A A5 02                   LDA   $0002
022C 85 05                   STA   $0005
022E A5 03                   LDA   $0003
0230 85 06                   STA   $0006
0232 A2 13                   LDXIM $13     LOAD DISPLAY POSITION POINTER
0234 8A            LOOP      TXA           PUT X VALUE INTO A
0235 48                      PHA           SAVE ON STACK
0236 A0 04                   LDYIM $04     SET TO SHIFT FOUR POSITIONS
0238 A5 04                   LDA   $0004   GET LEAST SIGN DIGIT REMAINING
023A 29 0F                   ANDIM $0F     MASK TO SINGLE CHARACTER
023C 18                      CLC           CLEAR
023D 69 30                   ADCIM $30     CONVERT 0-9 TO ASCII 0 - 9
023F 09 80                   ORAIM $80     BIT 80 MUST BE ON FOR AIM
0241 20 7B EF                JSR   $EF7B   AIM OUTPUT ROUTINE
0244 46 06         SHIFT     LSR   $0006   SHIFT TO GET HIGH HALF OF
0246 66 05                   ROR   $0005   DIGIT INTO POSITION
0248 66 04                   ROR   $0004
024A 88                      DEY           DECREMENT FOUR SHIFT COUNTER
024B D0 F7                   BNE   SHIFT   KEEP ON SHIFTING
024D 68                      PLA           RESTORE X FROM STACK
024E AA                      TAX
024F CA                      DEX           DECREMENT POSITION POINTER
0250 E0 0E                   CPXIM $0E     TEST 6 DIGITS OUTPUT
0252 B0 E0                   BCS   LOOP    MORE TO DO
0254 4C 26 02                JMP   DISPLY DONE. NOW START OVER AGAIN.
```

24 HOUR CLOCK INTERRUPT SERVICE

```
0300                  ORG   $0300

0300 48        INTRPT PHA         COME HERE ON TIMER INTERRUPT
0301 E6 00            INC   $0000 SAVE A REG AND BUMP COUNTER IN 0000
0303 D0 32            BNE   IDONE DONE WITH INTERRUPT
0305 38               SEC         SET DECIMAL MODE FOR CALCULATIONS
0306 18               CLC
0307 A5 01            LDA   $0001 BUMP ONE SECOND COUNTER
0309 69 01            ADCIM $01   BY ADDING 1 WITH CARRY
030B 85 01            STA   $0001 SAVE
030D C9 60            CMPIM $60   TEST SIXTY SECONDS
030F 90 22            BCC   NOTMIN NOT A MINUTE
0311 A9 00            LDAIM $00   A MINUTE
0313 85 01            STA   $0001 ZERO SECOND COUNTER
0315 18               CLC         THEN BUMP MINUTES
0316 A5 02            LDA   $0002 GET MINUTES COUNTER
0318 69 01            ADCIM $01   AND BUMP
031A 85 02            STA   $0002 SAVE
031C C9 60            CMPIM $60   TEST HOUR
031E 90 13            BCC   NOTMIN NOT AN HOUR YET.
0320 A9 00            LDAIM $00   AN HOUR, SO ZERO MINUTES
0322 85 02            STA   $0002
0324 18               CLC         THEN FIX HOURS
0325 A5 03            LDA   $0003
0327 69 01            ADCIM $01
0329 85 03            STA   $0003
032B C9 24            CMPIM $24   TEST 24 HOURS
032D 90 04            BCC   NOTMIN NOT 24 HOURS
032F A9 00            LDAIM $00   AT 24 HOURS RESET TO ZERO
0331 85 03            STA   $0003

0333 A9 EC     NOTMIN LDAIM $EC   RESET 20 INTERRUPT COUNTER
0335 85 00            STA   $0000

0337 AD 04 A0  IDONE  LDA   $A004 RESTART TIMER BY READING
033A D8               CLD         CLEAR DECIMAL MODE
033B 68               PLA         RESTORE A REGISTER
033C 40               RTI         RETURN FROM INTERRUPT
```

SUPERKIM
by MICROPRODUCTS



Here is a powerful microprocessor control system development tool and a complete microcomputer in one low-cost package. The Superkim singleboard computer has more features, more interface and expansion capability with a higher quality design and construction than any other in its class.

Have you got a thousand hours tied up in software for your *KIM-1 BASED control system and now your *KIM-1 is too small? The Superkim is the economical next step for expansion into more RAM, user EPROM and prototype area on one modern, compact, high density, fully assembled and integrated board. The Superkim has more software available than any other singleboard computer since it is totally compatible with KIM-1.

The Superkim has a wide range of appeal to engineers, teachers and industry.

* KIM-1 is a product of MOS Technology.

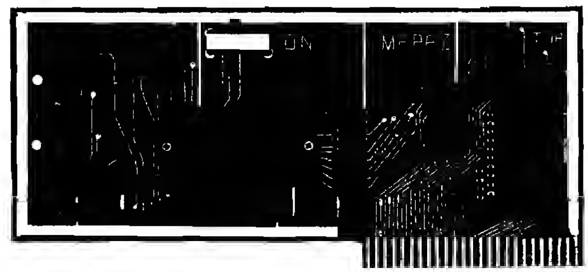
$445.00

Apple II EPROM
Expand Your ROM Software

Add capability to your system monitor or BASIC for business or other applications. Add commands to the operating system. New operating systems can be put into EPROM memory with our EPROM programmer and plugged directly into your APPLE II board with our EPROM socket adaptor. The **MICROPRODUCTS** EPROM Programmer will program INTEL 2716s, 2758s and other 5-volt replacements for 2716s.

Add to or replace existing APPLE II ROM software with operating systems of your own design. Other software systems similar to PASCAL, FORTH, LISP, APL, FORTRAN, COBOL, ALGOL, other BASIC's, etc. may be incorporated into your APPLE II ROM space.

The EPROM Programmer looks just like memory to the computer and can be configured to program memory locations from 8000 to FFFF for a total of 32K bytes. This means that the EPROMs can be used in computer applications other then the APPLE II, i.e. the **MICROPRODUCTS** Superkim, etc. This turns your APPLE II into a very low cost powerful, software development system.



FEATURES:
- Fully assembled.
- Completely self-contained
- Textool Zero insertion force socket for EPROM
- Onboard 25 volt power supply
- Double sided plated through holes on fiberglass PC board
- Gold plated edge connector
- Fully socketed
- Latest low-power Schottky IC's
- Solder mask

$99.95

MICROPRODUCTS will custom program EPROMs with your program on request

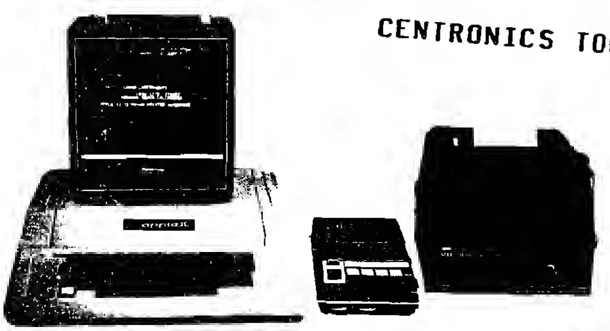
ADVANTAGES:
- Put memory in two empty ROM slots in APPLE II
- Replace memory in existing APPLE II ROM slots
- Add new operating systems to APPLE II
- Programs INTEL 2716 2K byte EPROM's, 2758 1K byte EPROMs and other compatible 5 volt EPROMs
- Put peripheral drivers in permanent memory
- Use APPLE II to program EPROMs for other computers

microproducts
2107 ARTESIA BOULEVARD
REDONDO BEACH, CA 90278
(213) 374-1673

Add a Printer to Your Apple II

With our PC Board that interfaces with the highly popular Southwest Technical Products PR-40 Printer. Both our Printer Interface and PR-40 Printer are available at computer stores.

CENTRONICS TOO!



Printer Interface Apple II to PR-40

Interface is completely assembled, tested and guaranteed. Including: Interconnecting cable, software stored on audio cassette, PC Board which plugs directly into your APPLE II

FEATURES:
- Prints one line at a time when return key is pressed.
- Ideal for writing programs, as you have a complete permanent record of all changes and deletions to your program.
- While in Basic, using the list mode, printer will list the entire program without stopping as the screen scrolls up one line at a time.
- You can refer to an earlier part of your program without the necessity of relisting it on the screen.
- Printer can be called from Basic to print entire contents of video screen.
- When using assembly language mode, one line at a time will be printed in the same format as the video screen.

$49.95

SPECIFICATIONS:
Interface hardware consists of:
- an epoxy fiberglass PC Board
- double-sided
- plated through holes
- silk screen printed legends
- gold plated edge card connector.

microproducts
2107 ARTESIA BOULEVARD
REDONDO BEACH, CA 90278
(213) 374-1673

microproducts
2107 ARTESIA BOULEVARD / REDONDO BEACH / CALIFORNIA 90278 / (213) 374-1673

MICROPRODUCTS/APPLE II

PARALLEL INTERFACE BROCHURE

The MICROPRODUCTS Parallel Output card allows your Apple II* computer to communicate with the outside world.

Applications include:

- Printer Interface
- Power controller
- Tone/Music generator
- Plotter Driver
- LED/LAMP Driver
- Apple II/Superkim Communications interface

Features:

- 8 bits output
- 15 ma output, current sink or source (Can drive L.E.D.s directly)
- TTL or CMOS compatible
- Will go in any slot on the Apple II*
- Data available strobe

General Information:

Data can be transferred to an external device by a STA, STY or STX from assembly language, or a POKE from BASIC. The 8 bits output can drive 2 7-segment L.E.D. displays, relays, SCRs, Printer, or anything which requires up to 8 bits of data.

Application notes and software to drive a Southwest Technical PR-40 printer is available for a nominal $5.00 extra.

Included with the Parallel Interface are instructions on how to interface to a Centronics 779 printer and wiring diagrams for use as a power controller. Additional software and applications notes available.

*APPLE II is a trademark of APPLE COMPUTER, INC.

$44.95

APPLE II - TRACE LIST UTILITY

Alan G. Hill
12092 Deerhorn Rd.
Cincinnati, OH 45240

Did you ever use the TRACE function in Integer BASIC, only to give up in despair after looking at a screeen full of line numbers? Try it without a printer and you may never use TRACE again! Well, here's the utility that will put TRACE back into your debugging repertoire (for those of us who need a little help getting it right.)

The utility presented here will list each BASIC program source statement line by line in the order executed. There's no need to refer back and forth between TRACE line numbers and the source program listing. Two versions are presented: Version 1 is a real-time utility; i.e. each statement is listed immediately prior to execution so you can follow the programs logical sequence. You can slow the execution rate down or even temporarily halt execution while you scan the screen. Version 2 only saves the line numbers of the last 100 lines executed for listing later. Version 3 could be useful in tracing a full-screen graphics program.

The Technique

The program utilizes the COUT hook at $36.37 to intercept and suppress TRACE printing. All other printing continues normally with one exception (see Warning #1). Before returning to the BASIC interpreter, the line number is picked up and pushed into an array (TR) in the variables area above LOMEM. If the number is the same as the previous line number, a zero line number is placed in the stack with the line number of a FOR I = 1 to 1000: NEXT I delay loop, for instance. When the number changes, it will be placed in the stack. The most recent 100 line numbers are saved. Tracing is performed under user control by the normal TRACE/NOTRACE statements. In Version 2, the lines may then be listed after the test program ends. The technique in Version 1 is similar with one distinction. The trace intercept routine transfers control to the utility program to list the line as soon as it is put in the stack.

How The TRACE Intercept Routine Works

The output pointer in $36.37 is initialized by the utility to the address ($300) of the Trace Intercept Routine. Each character is examined by TIR as it comes through if the TRACE flag is up (bit 7 of $AO on). If off, TIR jumps back to the normal print utility at $FDFO. If the character is a # ($A3), it is assumed that a line number follows. Every line number in the stack is pushed down and the current line number is placed at the top. Location $DC.DD points to the BASIC line about to be executed. The line number is in the second and third bytes. In Version 2, TIR returns to the interpreter. In the real-time version (Version 1), control is next transferred to the utility program at line 30020. TIR expects that the address of line 30010 has been saved in $15.16 by the utility programs CALL 945 in line 30010. TIR first saves the contents of $DC.DD and then replaces it with the contents of $15.16. It also saves the address of the current statement within the BASIC line. That is, the contents of $EO.E1 are saved at $1B.1C. TIR can now transfer control back to the interpreters continue entry point by a JMP $E88A which then executes line 30020 of the utility. The current line of the test program is listed; the BASIC pointers are restored by the CALL 954 in line 30090; the return address is popped; and control is returned to the test program through $E881. Fait accompli.

As mentioned previously, the TR array is used to save the line numbers. The array is set up the first time TIR is entered. Note that TR is intentionally not DIMensioned in the utility. TIR must handle that task since a RUN of the test program will reset the variables area pointer ($CC.CD) back to LOMEM

Programming The Routines

TIR starts at $300. It could be relocated if the absolute references in the POKE and CALL statements are changed. Also note that the LIST statement in lines 30060 and 32040 will not be accepted by the Syntax checker. They must first be coded as PRINT statements, located, and changed to LIST tokens ($74) using the monitor. This is more easily done if these lines are coded and the tokens changed before the remaining lines are entered. See example below for the case where HIMEM is 32768:

```
NEW
30060 PRINT EXECLINE
32040 PRINT TR (I)
(hit reset to enter monitor)
*7FEC;74
*7FF9:74
(enter Control/C)
LIST
30060 LIST EXECLINE
32040 LIST TR (I)
```

Using The Utility

1. After coding the assembler and BASIC utility programs, the test program is then appended. This may be done by a RUN 31000. Start the cassette recorder and hit Return when prompted. The test program will be appended to the utility program provided its highest line number is less than 29970.

2. Create a line O that will be used to indicate that a line has successively executed. For example, code:
   O REM ***ABOVE LINE REPEATED***

3. **Run** the utility of your choice:
   RUN 30000 Version 1 (Real-time list)
   or RUN 32000 Version 2 (Post-execution list)

4. Insert the TRACE/NOTRACE statements wherever desired in test program. Just enter the TRACE command directly if you want to trace the entire program. Also see Warning #1.

5. RUN the test program.

6. Display the results:
   A. **Real-time Version:** The lines will be listed automatically as executed. Note the FOR: NEXT loop in line 30090 can be adjusted to control the execution rate. The upper limit could be PDL(O), thereby giving you run-time control over the execution rate. Note also that execution can be forced to pause by depressing paddle switch O. Execution will resume when the switch is released.

B. **Post-execution Version:** After stopping or ending the program, enter a GOTO 32020 command. The first page of statements will be displayed. Enter a "C" to display additional pages, a "T" to reset for another test run, or an "E" to return to BASIC. Note that even if you have traced with Version I, you can still display the last 100 lines with Version 2.

Sample Run

Test Program

```
0     REM *** REPEATED ***
10    TRACE
30    GOSUB 100+RND(3) *10
40    FOR I=1 TO 10: NEXT I
50    GOTO 30
100   PRINT "LINE 100":RETURN
110   PRINT "LINE 110":RETURN
120   PRINT "LINE 120":POP
125   NO TRACE:END
>     RUN 30000
>     RUN
```

Trace Output

```
    30    GOSUB 100+RND(3)*10
    110   PRINT "LINE 110":RETURN
LINE 110
    30    GOSUB 100+RND(3)*10
    40    FOR I=1 TO 10:NEXT I
    0     REM *** REPEATED ***
    50    GOTO 30
    30    GOSUB 100+RND(3)*10
    120   PRINT "LINE 120":POP
LINE 120
    125   NO TRACE:END

    >
```

For a slow motion game of "BREAKOUT", trace it with the real-time version!

Hints And Warnings

It's usually a good idea to deactivate TIR after the test program has ended by hitting Reset and Control/C and entering NOTRACE. Don't try to trace the test program without first running the utility program at line 30000 or 32000.

To increase the debugging power of the real-time trace utility, make liberal use of the push button to halt program execution. With practice and the proper choice of the delay loop limit in line 30090, you can step through the program one line at a time. Enter a Control/C while the push button is depressed and execution will be STOPPED AT 30070. You can then use the direct BASIC commands to PRINT and change the current value of the programs variables. Enter CON and execution will resume.

With additional logic in the utility program, you can create specialized tracing such as stopping after a specified sequence of statements has been detected. Return via a CALL 958 if you don't want TRACE turned back on.

Tracing understandably shows the execution rate of your program, but you probably aren't concerned with speed at this point. However, the wise use of TRACE/NOTRACE will help move things along. Also, when encountering a delay loop such as FOR I=1 to 3000: NEXT I, you may want to help it along by stopping with a Control/C entering I=2999, and CONtinuing.

Warning #1: There must be **no** PRINT statement with a # character in the output. TIR assumes that a # is the beginning of a trace sequence. Either remove the # or bracket the PRINT statement with a NOTRACE/TRACE pair.

Warning #2: There must be no variable names in the test program identical to those in Version 1. The TR variable name must be unique in both versions.

Warning # 3: Line O in the test program should be a REMark statement as described above to avoid confusion. Line O is listed when a line is successively repeated.

Warning # 4: Once TRACE has been enabled, the test program must not dynamically reset the variables pointer ($CC.CD) with a CLR or POKE unless it first disables TRACE and resets $13.14; e.g. 100 NOTRACE:CLR: POKE 19, O: POKE 20,0: TRACE is OK.

Extensions

The primary motivation for this program was to improve the TRACE function in Integer BASIC. However, one can imagine other uses of a program that gains control as each statement is executed — maybe the kernel of a multiprogramming executive. I would be interested in seeing your comments and modifications.

ZERO PAGE MEMORY MAP

Location	Use
$00.01	SAVE AREA FOR HIMEM. APPEND USES
$05	PROGRAM SWITCH ON=$FF OFF=$7F Turned on when trace # character ($a3) is detected. Turned off when next space character ($AO) is detected
$13.14	ADDRESS OF TR STORAGE VARIABLE
$15.16	ADDRESS THAT CAUSES RETURN TO LINE 30020 IN BASIC LIST UTILITY (Version I)
$17.18	SAVE AREA FOR $DC.DD. ADDRESS OF CURRENT BASIC LINE IN TEST PROGRAM
$1B.1C	SAVE AREA FOR $EO.E1. ADDRESS OF STATEMENT WITHIN BASIC LINE
$AO	APPLE II TRACE FLAG ON=$FF OFF=$7F

TRACE INTERRUPT ROUTINE

BY ALAN G. HILL
23 NOVEMBER 1978


```
0300                  ORG   $0300

0300 24 A0     START  BIT   $00A0  IS TRACE ON?
0302 30 03            BMI   $0307  BRANCH YES
0304 4C FD FD  PRINT  JMP   $FDFD  NO. BACK TO PRINT
0307 C9 A3     TRACE  CMPIM $A3    NUMBER SIGN?
0309 F0 0D            BEQ   SWON   BRANCH YES. IT'S A TRACE LINE
030B 24 05            BIT   $0005  SWITCH ON?
030D 10 F5            BPL   PRINT  NO. PRINT CHARACTER
030F C9 A0            CMPIM $A0    SPACE?
0311 D0 04            BNE   RETURN NO. RETURN W/O PRINTING
0313 4C D3 03         JMP   TRCOFF VER. II LDAIM $7F
0316 EA               NOP          VER. II STA $05
0317 60        RETURN RTS          BACK TO BASIC

0318 A9 FF     SWON   LDAIM $FF    TURN ON SWITCH
031A 85 05            STA   $0005
031C A5 13            LDA   $0013  FIRST TIME THRU?
031E D0 49            BNE   GLINO  BRANCH NO. TO GET LINE NO.
0320 A5 14            LDA   $0014
0322 D0 45            BNE   GLINO
0324 A5 CD            LDA   $00CD  YES. SETUP TR ARRAY
0326 85 14            STA   $0014  IN VARIABLES
0328 A5 CC            LDA   $00CC  AREA AND ADJUST
032A 85 13            STA   $0013  POINTER
032C 18               CLC
032D 69 0F            ADCIM $0F
032F 85 CC            STA   $00CC  NEW PV
0331 A5 CD            LDA   $00CD
0333 69 00            ADCIM $00
0335 85 CD            STA   $00CD
0337 A0 00            LDYIM $00
0339 A9 D4            LDAIM $D4    "T"
033B 91 13            STAIY $13
033D C8               INY
033E A9 D2            LDAIM $D2    "R"
0340 91 13            STAIY $13
0342 C8               INY
0343 A9 00            LDAIM $00
0345 91 13            STAIY $13    DSP
0347 C8               INY
0348 A5 CC            LDA   $00CC
034A 91 13            STAIY $13    NVA
034C C8               INY
034D A5 CD            LDA   $00CD
034F 91 13            STAIY $13
0351 18               CLC
0352 A9 04            LDAIM $04    POINT $13.14 TO TR
0354 65 13            ADC   $0013  DATA AREA-1
```

```
0356 85 13          STA   $0013
0358 A5 14          LDA   $0014
035A 69 00          ADCIM $00
035C 85 14          STA   $0014
035E A0 CA          LDYIM $CA    INITIALIZE TR ARRAY
0360 A9 FF          LDAIM $FF    TO ALL FF'S
0362 91 13    FLOOP STAIY $13
0364 88             DEY
0365 D0 FB          BNE   FLOOP  LOOP TIL DONE
0367 F0 29          BEQ   SLINE  ALWAYS

0369 A0 02    GLINO LDYIM $02
036B B1 13    TLINE LDAIY $13    IS LAST LINE NO.
036D D1 DC          CMPIY $DC    SAME AS THIS ONE?
036F D0 08          BNE   NLINE  BRANCH NO
0371 88             DEY
0372 D0 F7          BNE   TLINE  LOOP
0374 98             TYA          YES. PUT ZERO
0375 48             PHA          LINE NO. IN
0376 48             PHA          STACK TEMPORARILY
0377 F0 21          BEQ   TSTACK ALWAYS

0379 A0 02    NLINE LDYIM $02    IS THERE ALREADY A
037B B1 13    TLOOP LDAIY $13    ZERO AT THE TOP?
037D D0 13          BNE   SLINE  BRANCH NO TO GET LINE NO.
037F 88             DEY
0380 D0 F9          BNE   TLOOP  LOOP
0382 A2 02          LDXIM $02    YES
0384 C8             INY
0385 B1 DC    CLOOP LDAIY $DC    COMPARE WITH NEXT
0387 C8             INY          LAST LINE NO.
0388 C8             INY
0389 D1 13          CMPIY $13
038B D0 05          BNE   SLINE  IT'S DIFFERENT. SAVE IT
038D 88             DEY          IT'S SAME
038E CA             DEX
038F D0 F4          BNE   CLOOP  LOOP
0391 60             RTS          STILL THE SAME.  RETURN TO TRACE

0392 A0 02    SLINE LDYIM $02
0394 B1 DC    PLINE LDAIY $DC    PICK UP LINE NO.
0396 48             PHA          HOLD IN STACK TEMPORARILY
0397 88             DEY
0398 D0 FA          BNE   PLINE  BOTH DIGITS
039A A0 CB   TSTACK LDYIM $CB    PUSH DOWN ALL TR
039C B1 13    PLOOP LDAIY $13    ELEMENTS TO
039E C8             INY          MAKE ROOM FOR
039F C8             INY          NEW LINE NO. AT TR(0)
03A0 91 13          STAIY $13
03A2 88             DEY
```

```
03A3 88             DEY
03A4 88             DEY
03A5 D0 F5          BNE    PLOOP  LOOP UNTIL DONE
03A7 A0 01          LDYIM  $01
03A9 68             PLA           PUT NEW LINE NO. OR
03AA 91 13          STAIY  $13    ZERO IN TR(0)


03AC C8             INY
03AD 68             PLA           GET HIGH ORDER BYTE
03AE 91 13          STAIY  $13    STUFF IT TOO
03B0 60             RTS           RETURN TO BASIC

03B1 A5 DC   SAVE   LDA    $00DC  ROUTINE TO SAVE ADDRESS
03B3 85 15          STA    $0015  SO TIR WILL CAUSE BASIC
03B5 A5 DD          LDA    $00DD  TO EXECUTE LINE 30020
03B7 85 16          STA    $0016  WHEN TRACE SEQUENCE IS DETECTED
03B9 60             RTS           RETURN TO UTILITY

03BA A9 FF   TEST   LDAIM  $FF    ROUTINE TO RE-ENTER TEST PGM
03BC 85 A0          STA    $00A0  TURN TRACE BACK ON
03BE A5 17          LDA    $0017  RESTORE TEST PROGRAM
03C0 85 DC          STA    $00DC  LINE NO.
03C2 A5 18          LDA    $0018
03C4 85 DD          STA    $00DD  AND
03C6 A5 1B          LDA    $001B
03C8 85 E0          STA    $00E0  STATEMENT ADDRESS
03CA A5 1C          LDA    $001C
03CC 85 E1          STA    $00E1
03CE 68             PLA           POP UTILITY ADDRESS
03CF 68             PLA           FROM STACK
03D0 4C 81 E8       JMP    $E881  RE-ENTER BASIC TRACE ROUTINE

03D3 A9 7F   TRCOFF LDAIM  $7F    TURN OFF
03D5 85 C5          STA    $00C5  SWITCH AND
03D7 85 A0          STA    $00A0  TRACE: (DON'T TRACE UTILITY)
03D9 A5 DC          LDA    $00DC
03DB 85 17          STA    $0017  SAVE ADDRESS OF
03DD A5 DD          LDA    $00DD  TEST PGM LINE NO.
03DF 85 18          STA    $0018
03E1 A5 15          LDA    $0015  SETUP TO GO TO UTILITY
03E3 85 DC          STA    $00DC  TO LIST LINE NO.
03E5 A5 16          LDA    $0016  SETUP LINE ADDRESS
03E7 85 DD          STA    $00DD
03E9 A5 E0          LDA    $00E0  SETUP STATEMENT ADDRESS
03EB 85 1B          STA    $001B
03ED A5 E1          LDA    $00E1
03EF 85 1C          STA    $001C
03F1 68             PLA           REMOVE ADDRESS FROM STACK
03F2 68             PLA
03F3 4C 8A E8       JMP    $E88A  GO TO UTILITY VIA CONTINUE
```

VERSION I: Real-Time Trace List Utility Program

```
29770  REM REAL-TIME TRACE LIST UTILITY PROGRAM
29980  REM SET-UP COUT AND INITIALIZE ZERO PAGE VALUES
29990  REM SET-UP TIR ASSEMBLER JUMP
30000  NOTRACE; POKE 54,768 MOD 256: POKE 55,768/256:
       POKE 19,0:POKE20,0=POKE 787,76: POKE 788,211:
       POKE 789,3: POKE 790,234
30005  REM SAVE ADDRESS SO TIR RETURNS TO LINE 30020
30010  CALL 945 :END
30020  EXECLINE=TR(0): IF EXECLINE #0 THEN 30050
30030  IF RRRRR=1 THEN 30070
30040  RRRRR=1: GOTO 30060
30050  RRRRR=0
30060  LIST EXECLINE
30070  IF PEEK (-16287)>127 THEN 30070: REM PAUSE IF SW(0) ON
30080  IF EXECLINE = 0 THEN 30100: REM SKIP DELAY
30090  FOR JJJJJ=1 TO 100: NEXT JJJJJ: REM DELAY
30100  CALL 954: REM BACK TO TEST PGM
30110  END: REM NEVER EXECUTED

31000  REM APPEND TEST PROGRAM
31010  INPUT "HIT RETURN TO APPEND" A$
31020  POKE 0, PEEK(76): POKE 1, PEEK (77): POKE 76, PEEK (202):
       POKE 77, PEEK (203): CALL-3873: POKE 76, PEEK (0):
       POKE 77, PEEK (1):END
```

VERSION II: Post-Execution Trace List Utility Program

```
32000  NOTRACE: POKE 54,768 MOD 256: POKE 55,768/256: POKE 19,0:
       POKE 20,0: POKE 787,169: POKE 788,127:
       POKE 789,133: POKE 790,5
32010  PRINT "TRACE SET UP. ENABLE TRACE IN TEST PROGRAM": END
32015  REM GOTO 32020 WHEN TEST PGM ENDED
32020  NOTRACE: POKE 54,240: POKE 55,253:
       IF PEEK (20)#0 THEN 32030: PRINT "TRACE
       WAS NOT ON IN TEST PROGRAM": GOTO 32090
32030  CALL-936: FOR I=100 TO 1 STEP-1:
       IF TR (I)=-1 THEN 32060
32040  LIST TR (I)
32050  IF PEEK (37)>18 THEN 32090
32060  NEXT I
32070  GOTO 32090
32080  CALL-936: IF I>1 THEN 32060
32090  PRINT:PRINT "C/T/E?"
32100  KEY=PEEK(-16384): IF KEY< 128 THEN 32100:
       POKE-16368,0: IF KEY=212 THEN 32000:
       IF KEY=195 THEN 32080:END
```

THE MICRO SOFTWARE CATALOG: VI

Mike Rowe
P.O. Box 3
S. Chelmsford, MA 01824

Name: **MAXIT!**
System: **PET**
Memory: **8K**
Language: **BASIC**
Hardware: **Standard**
Description: A challenging number game played between two persons or versus the PET. From an 8 × 8 board players alternatively move horizontally and vertically trying to maximize their score and minimize their opponents. An exciting, engrossing game, that bears returning to multiple times. Suitable for young and old alike. Excellent graphics.
Copies: **50 plus**
Price: **$4.95** plus 32¢ tax for CA residents, pp.
Includes: Cassette and 2 page printed instructions.
Author: **Harry J. Saal**
Available from:
Harry J. Saal
810 Garland Drive
Palo Alto, CA 94303

Name: **6502 Tiny Editor - Assembler**
System: **Any 6502 based system.**
Memory **Program takes 1K, 4K** recommended for source and object code and label table.
Language: **Machine Language**
Hardware: **ASCII Keyboard** and **CRT** display.
Description: A single pass assembler, closely follows MOS Mnemonics, and is extremely memory efficient. The editor is designed to be easily extended by the user. Editor commands include: Find line, delete line, insert line, list source, list symbolic labels, define label, and set origin. A single pass assembler allows the object code to overwrite the source code - larger source programs can be assembled in a given memory size.
Copies: **Just released:**
Price: **$19.95** (KIM-1 Hypertape cassette: $3.00 extra)
Include: User manual and complete source and object listing, fully commented, with modification instructions.
Author: **Michael Allen**
Available from:
Michael Allen
6025 Kimbark
Chicago, IL. 60637

Name: **6502 ROBOT**
System: **Any 6502 based system**
Memory: **1.5K**
Language: **Machine language**
Hardware: **ASCII Keyboard** and **CRT** display, or "turtle", or plotter.
Description: ROBOT is an interactive programming language for the control of robots, such as "turtle", plotter or CRT cursor. ROBOT's command processing module is designed to allow the user to design his own language of personalized commands and command subroutines to suit his particular application. The version offered here includes a command set and subroutine package for the control of a CRT robot.
Copies: **Just released.**
Price: **$5.00** (KIM-1 Hypertape cassette: **$3.00** extra)
Include: user manual, complete and fully commented source and object listing, instructions for adapting, modifying, and using the command processing module for other applications.
Author: **Michael Allen**
Available from:
Michael Allen
6025 Kimbark
Chicago, IL 60637

Name: **OSI Games**
System: **Challenger**
Memory: **4K 8K**
Language: **Basic and Assembly**
Hardware: **Challenger**
Description: The game programs are written for the challenger with the 440 video display and ASCII keyboard. Most of these will run on the 2p and 1p. Games such as Bomber and Klingon are written with simulated animation and Klingon also will support sound with PIA port and tone oscillator. We also have lunar lander; Battleship; and others.
Copies: **Just released**
Price: **$8.00** for listing and instructions and 300 baud cassette
Author: **William L. Taylor**
Available from:
William L. Taylor
264 Flora Rd.
Leavittsburg, Ohio 44430

Name: **LINK**
System: **PET**
Memory: **Any amount**
Language: **Assembly**
Hardware: **Standard PET**
Description: This program will allow the user to link exclusively numbered BASIC programs in memory. This allows the programmer to develop complex programs as sub modules and then merge them together into the final functioning unit. A great time saver as the programmer can develop a library of subroutines which can be merged virtually at any time with the program which he is developing. With complete instructions on use.
Copies: **Just released**
price: **$12.95** ppd, Michigan residents add 4 % sales tax.
Includes: **Cassette and instructions**
Author: **G. Salked**
Order Info.: Master Charge and Visa accepted.
Available from:
Your local PET dealer or
Dr. Daley
425 Grove Ave.
Berrien Springs, MI 49103
616-471-5514

Name: **PILOT**
System: **PET**
Memory: **8K minimum**
Language: **BASIC**
Hardware: **Standard PET**
Description: A simple to use, easy to learn programming language. This is especially suited for use by children. Only 10 commands to learn with no complicated syntax plus special cursor and graphics control commands.
Copies: **25**
Price: **$12.95** ppd, Michigan residents add 4% sales tax.
Includes: cassette and users manual.
Author: **R.F. DAley**
Orther Info.: **Master Charge and Visa accepted**
Available from:
Your local PET dealer or
Dr. Daley
425 Grove Ave.
Berrien Springs, MI 49103
616-471-5514

Name: **BASIC Modification Package**
System: **KIM** expanded to run Microsoft-9 digit KIM BASIC
Memory: **Locations DD to E0 and 200 to 2E4** used in addition to locations in unmodified program.
Language: **Machine**
Hardware: **None additional.** Optionally supports a terminal with x-on/x-off feature.
Description: Enhancements and modifications to Microsoft 9-digit KIM BASIC (sold by Johnson Computer). Machine Language patches to original program. BASIC and mods can be loaded with only one tape. Jim Butterfield's Hypertape (and other routines) are relocated to low memory on initialization. SAVE and LOAD at Hypertape speeds. SAVE and LOAD messages improved. SAVE returns to BASIC. Programs with higher line numbers can be appended. This means BASIC subroutines, DATA statements and utility programs (RENUMBER) can be added after program development. Interrupt running programs and listing reliably with ST button. GET (one character or digit) command noted and fixed. Terminals with x-on/x-off feature will load paper or cassetts tapes perfectly. BASIC programs saved on cassette tapes with different initialization conditions can be used interchangably. A 1/10 sec counter can be started, stopped and read under program control. Time and control external events with this jeffrey counter (named after former student and pun intended).
Copies: **> 10**
Price: **$15** check or money order.
Includes: Object code listing, instructions, examples, miscellaneous information and help from the author (by correspondence).
Author: **Harvey B. Herman**
Available from:
Harvey B. Herman
2512 Berkley Place
Greensboro NC 27403

Name: **PET Library**
System: **PET**
Memory: **8K**
Language: **Basic, some Assembler**
Hardware: **No Special**
Description: A variety of PET programs including games, educational, music, astronomy, financial, and many others
Copies: **100+**
Price **$2.50** first program **$1.50 each additional.**
Includes: **Cassette & Postage**
Order Info.: Send Business envelope and postage for complete list of programs available.
Author: **Russell Grokett**
Available from:
PET Library
401 Monument Rd. #177
Jacksonville, FLA 32211

Name: **LIFE for the KIM-1**
System: **KIM-1** with an **XITEX** VIDEO BOARD.
Memory: **2K** ($2000-$2800 plus 30 bytes on page zero.)
Language: **Assembler**
Description: This program will play Conway's game of LIFE. The program will plant one living cell in mid-screen, and then ask for coordinates, measured from the center, for other living cells. A generation takes about 1/6 second for every birth and death. The program may be patched to accommodate other video boards.
Copies: **Just released.**
Price: **$2.00** for description and listing.
$5.00 for object tape on cassette in HYPERTAPE format.
Author: **Theodore E. Bridge**
Available from:
Theodore E. Bridge
54 Williamsburg Dr.
Springfield, MA 01108

6522 CHIP SETUP TIME

John T. Kosinski
4 Crestview Drive
Millis, MA 02054

Richard F. Suitor
166 Tremont Street
Newton, MA 02158

MICRO 6:4 summarized some discussion from EDN concerning their difficulties with interface design. One point in particular caught our eye - a statement that the 6522 VIA chip cannot use the Apple-generated device select signal (from pin 41 of the I/O slot) because the data sheets clearly require that the chip be selected 180 ns before the I/O enable signal goes high, whereas the Apple-generated signals occur nearly simultaneously. That is a misconception which we would like to correct. We report a 6522 interface that uses the pin 41 select signal, that theoretically ought to work and in fact does work.

The 6522 VIA - Why Bother?

Since there are several interfaces both supplied by Apple and by other vendors, why bother? VIA stands for Versatile Interface Adapter. It was designed by MOS Technologies, the same folks who brought us the 6502 and it is well named. It has two I/O ports, two timers and a shift register, and so many options in operating them that we won't try to list them. A very useful feature is that all of the functions can interrupt the 6502. Several software tasks (cassette I/O, music, software generated serial I/O) require the Apple to spend most of its time in timing loops. With the use of timers and interrupts, these functions can be performed while the system is running some other program. You can have your STAR WARS theme while shooting TIE fighters, instead of after; more prosaically, you can print edited text while editing more. The 6522 is quite flexible because of its versatility; it is a definite asset to the Apple.

What's the Big Problem?

The 6522 was designed to work well with the 6502. The signals at the Apple I/O slots are not all 6502 signals, however - some are decoded device select signals, which would be very convenient to use if we could. According to the referenced letter, we can't - there is not enough time to select the chip. As mentioned before, the problem is not insurmountable; let's discuss timing a bit. The 6522 has 16 registers that control all the bells and whistles. To communicate with the 6522 from the CPU, one:

1. Selects one of the 16 registers with the address lines.
2. Selects (turns on) the 6522 chip itself.
3. Enables the I/O transaction.
4. Disables the I/O transaction.
5. De-selects the chip.

Some of the processes take time. For example, th 6522 data sheets DO say that the address must be valid 180 ns before the I/O enable. They ALSO state that the select is normally derived from the address lines. However, the timing tolerance referred to is the register select operation of step 1, and it must occur 180 ns before the I/O enable of step 3. The data sheets DO NOT specify the chip select time of Step 2. A representative of MOS Technologies, looking at the circuit diagrams, estimated that it would be sufficient to have Step 2 occur 40 - 50 ns before Step 3. He did not offer a minimum lead time requirement.

The 6502 and the 6522 expect that Step 3 will occur when the 6502 02 signal goes high and that Step 4 will occur when 02 goes low. The enable signal presented at the I/O port of the Apple is actually ɸ0, a signal which leads 02 by 50 - 70 ns. That is a very short time, but long compared to the 10 ns or so it takes an LS gate to operate. There are three LS gates involved in a transfer (the chip itself, and data bus buffers at each end) giving a nominal 30 ns timing tolerance. Actually, if the devices on the data bus are properly tristated (i.e. they have very high impedance unless they are active), the capacitance of the bus and the buffer delays will probably permit proper operation with the ɸ0 enable pulse. There certainly seem to be several circuits using that signal that work (now including, for some unknown reason, EDN's.)

In summary, there are perhaps two problems in interfacing a 6522 to the Apple:

1. One may indeed need to select the chip before enabling the I/O, but no more than 40 - 50 ns before.
2. One may need to use an I/O enable signal that is coincident (within about 30 ns) with the 6502 02.

It is not at all clear what one could get away with if one tried; it is clear that if the requirements 1 and 2 are met, the 6522 should interface easily to the Apple II. However, since the device select and I/O select signals that Apple supplies de-select at the end of ɸ0, one should reasonably expect that an interface that tristates when these signals deselect should work satisfactorily with the Apple despite the fact that the 6502 is accepting data for another 50 ns. It is apparent from the discussion that has resulted from EDN's efforts that many interfaces so designed do work satisfactorily; it is not clear how marginal the operation is.

There is an interesting discussion of the Apple timing in the Sept. issue of KILOBAUD starting on page 10. They reported on a 6522 interface and found that the important time was the rise of the I/O enable signal. Since they do not mention what was done for chip select and for data bus buffering, one can only wonder if chip select timing was affecting their results.

We decided to play safe and satisfy both requirements. One way to satisfy the second is to use the real 02. As it turns out, this also satisfies the first, because 02 lags the device select signal by about 50 ns. This coincidence may have led to some confusion in interpreting timing experiments! This is the approach we followed; in retrospect, knowing what we do now, we would have proceded otherwise (i.e. perhaps used a delayed device select signal as an I/O enable signal.) Since it does no good to have the I/O enabled if the chip and the data bus buffers aren't, we lenghtened the device select signal by delaying it and ANDing it with itself. We had no problems with this approach. (It is not a 'better' solution than Mr. Scouten's; he is quite right that one cannot use both the pin 41 signal and the ɸ0 directly with the 6522 for their intended functions. The difference, however, between 180 and 50 ns required setup time makes it feasible to use the pin 41 decoded device select signal if one chooses.)

HIGH-RESOLUTION PLOTTING FOR THE PET

John R. Sherburne
206 Goddard
White Sands Missile Range, NM 88002

The PET Machine Language Monitor gives PET users a greatly expanded ability to devleop and use assembly language programs. While early buyers of PET have had to wait a while for the Monitor, the ability to save and load machine language programs directly to and from cassette is well worth the wait. Access to machine language has always been available through the POKE command, but translating op codes and addresses from hex to decimal and back is tedious. Also, the need to load a program via another BASIC program or via the keyboard is wasteful and time-consuming. PET's Monitor allows an assembly language program to be saved and loaded as easily as the BASIC program. Better yet, an assembly language program can be written to reside in an unused section of memory such as the second cassette buffer. A BASIC program can then be loaded in the usual manner and can use the machine language program as a subroutine.

One way that the use of a resident machine language routine can be a big help is in implementing high-resolution plotting on the PET. High-resoltuion plotting, in effect, expands PET's 40 × 25 character display to 80 × 50. To do so, each character is divided into quarter characters. The four basic quarter characters are displayed by pressing "SHIFT" and "," or ";" or "<" or ">". There are a total of sixteen possible combinations of these four quarter characters which can be used to produce a high-resolution plot. The process of producing such a plot in BASIC, however, is complex and slow. A machine language subroutine, on the other hand, can make the plotting process quite simple. For example, the Lissajous figure in * Figure 1 was plotted with this program:

```
10 POKE 1,58:POKE 2,3:PRINT (clr)"
20 DELTA=2*π/900
30 P=3:Q=4
40 FOR I=0 TO 900
50 THETA=DELTA*I
60 X=INT(39.5+38*COS(P*THETA))
70 Y=INT(25.5+24*SIN(Q*THETA))
80 POKE 81,X:POKE 82,Y:A=USR(0)
90 NEXT I
100 GET A$:IF A$="" THEN 100
```

The machine language routine is called in line 80 with the USR command after first POKEing the X and Y coordinates to be plotted in memory locations 81 and 82, respectively. The values of P and Q in line 30 determine the shape of the figure. The machine language plotting routine used by the program is listed below. The procedures for using it are:

LOADING - The program is initially loaded into the second cassette buffer beginning in location $033A using the Monitor. The program is saved on cassette with the command: .S,01,HI-RES,033A,03CA. The value $03CA is the endng address plus one. Once saved, the program can be reloaded into the cassette buffer with the normal command: LOAD"HI-RES".

BASIC INTERFACE - With HI-RES loaded, the BASIC driver program can be loaded from cassette using normal procedures or the "NEW" command can be given and a new BASIC program entered from the keyboard. Before HI-RES can be called, the starting address, $033A, must be entered in memory locations 0001 and 0002. This was done in line 10 of the program above. HI-RES can now be called by the USR command. Before each call, the X and Y coordinates must be POKED into decimal addresses 81 and 82, respectively. Valid coordinate values run from 0 to 79 in the X direction and from 0 to 49 in the Y direction. Position 0, 0 is in the upper left-hand corner of the screen.

OTHER - If zero is used as the argument of the USR command, the plotting routine will overwrite any character already on the screen. If a value other than zero is used any non-plot character already on the screen will be left there. Thus axes and text can be preprinted on the screen and a graph later plotted without distrubing the preprinted data.

RECREATIONAL GRAPHICS FOR PET

There are probably a lot of practical uses for the PET high-resolution graphics program described above but I haven't had time to find them yet. Instead, I have spent countless hours in front of the display watching PET draw intriguing designs for which there is relatively little practical purpose. My addiction started simply enough. To test the HI-RES plotting routine, I wrote a program to draw an elipse using the formula: X=P*COS(Θ); Y=Q*SIN(Θ). Pleased with the result, I added a FOR loop to vary the values of P and Q and produced the family of elipses shown in Figure 1. I didn't realize it but I had embarked on a project which would take every free moment for the next two weeks.

The next step was to modify the formula so that a flower rather than an elipse was produced. The new formula was:

X=R*COS(θ); Y=R*SIN(θ) where R=SIN(N*θ) If N is odd, a flower with N leaves is produced; if N is even, the flower will have 2N leaves. Figure 2A is an eight leaved flower using the formula R=SIN(4*θ). Figure 2B uses an alternate forumul: R=COS(4*θ). As with the ellipse, the next step was to produce a family of flowers (Figure 2C) by adding a FOR loop to vary the size of the flower and to alternate between the two formulas.

By now I was completely hooked. I dug into a dusty book of mathematical formulas and found two rather obscure figures, the epicycloid and hypocycloid. Best known from the toy "Spirograph", the epicycloid is formed by tracing the path of a point on the circumference of a circle as it is rolled around the outside of a second circle. The hypocycloid is formed when one circle is rolled around the inside of the other. The formulas are:

Epicycloid:

```
X=(P+Q)*COS(AN)+Q*COS(P+Q)*AN/Q)
Y=(P+Q)*SIN(AN)+Q*SIN(P+Q)*AN/Q)
```

Hypocycloid:

```
X=(P-Q)*COS(AN)+Q*COS(P-Q)*AN/Q)
Y=(P-Q)*SIN(AN)-Q*SIN(P-Q)*AN/Q)
```

*Note: Figure 1 on cover

In both formulas P represents the radius of the stationary circle and Q the radius of the rolling circle. A typical epicycloid is shown in Figure 3. To plot these more complex figures a minor technical problem had to be solved. Many of the larger "cycloids" require more than one revolution of the rolling circle around the stationary circle. To avoid either stopping too soon or running too long, I had to add a routine to compute the number of revolutions required for the full figure. Since the rolling circle makes P/Q Revolutions in one circuit of the stationary circle, a complete figure is made when the rolling circle turns the number of times equal to the first integer multiple of P/Q. That multiple, N, times 2 is the number of points or cusps in the cycloid. For convenience I print the number of cusps in the corner of the display. An eight cusp hypocycloid is shown in Figure 4. With both types of cycloid P and Q can be varied to produce a variety of figures. To avoid creating a figure too large to display, P must be ≤ 24 for a hypocycloid and $P+2*Q \leq 24$ for an epicycloid.

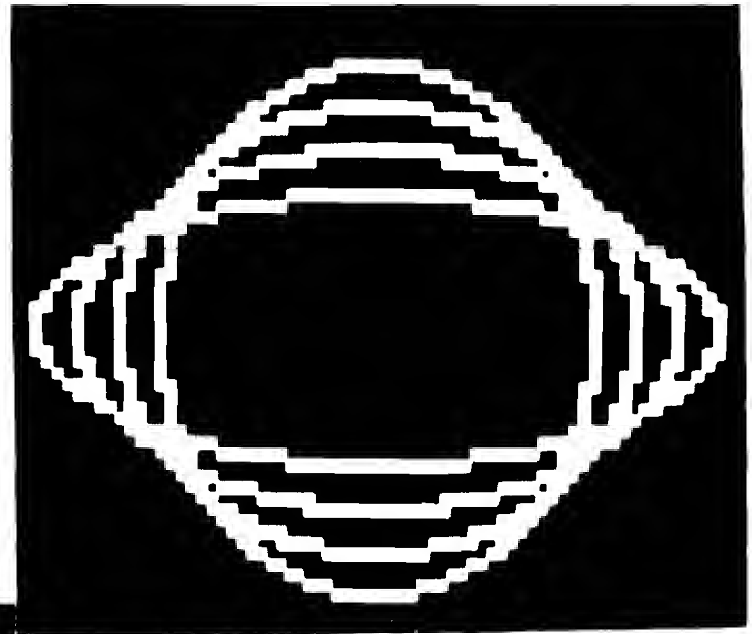
As a final fillip, a third parameter can be added to the cycloid programs. Rather than trace a point on the circumference of the rolling circle, a point at a distance R from the center of the circle is traced. The value of R can be larger or smaller than Q. If R is larger than Q the formulas for determining the largest figure which the display can accomodate are: epicycloid, $P+Q+R \leq 24$; hypocycloid, $P+R-Q \leq 24$.

```
                        HI-RESOLUTION

                        BY JOHN R. SHERBURNE
                        FEBRUARY 1979

033A                          ORG   $033A

033A A9 00      START   LDAIM $00    INITIALIZE
033C 85 53              STA   $0053
033E 85 56              STA   $0056
0340 38                 SEC
0341 A5 51              LDA   $0051
0343 E9 4F              SBCIM $4F    CHECK FOR VALID X
0345 30 03              BMI   CHECK
0347 E6 54              INC   $0054
0349 60                 RTS

034A 38         CHECK   SEC          CHECK FOR VALID Y
034B A5 52              LDA   $0052
034D E9 31              SBCIM $31
034F 30 03              BMI   HALF
0351 E6 55              INC   $0055
0353 60                 RTS

0354 46 51      HALF    LSR   $0051
0356 90 02              BCC   NOCAR
0358 E6 56              INC   $0056
035A 46 52      NOCAR   LSR   $0052
035C 90 04              BCC   NOCRY
035E E6 56              INC   $0056
0360 E6 56              INC   $0056  DIVIDE X AND Y BY 2
0362 A9 01      NOCRY   LDAIM $01    DETERMINE QUADRANT OF NEW POINT
0364 A4 56      LOOP    LDY   $0056  AND PLACE QUADRANT NUMBER IN $0056
0366 F0 06              BEQ   MATCH
0368 0A                 ASLA
0369 C6 56              DEC   $0056
036B 4C 64 03           JMP   LOOP
```

```
036E 85 56     MATCH  STA    $0056
0370 06 52            ASL    $0052
0372 06 52            ASL    $0052
0374 06 52            ASL    $0052
0376 A5 52            LDA    $0052
0378 06 52            ASL    $0052
037A 26 53            ROL    $0053  MULTIPLY Y BY DECIMAL 40.
037C 06 52            ASL    $0052  (NO. CHARACTERS PER LINE)
037E 26 53            ROL    $0053
0380 65 52            ADC    $0052
0382 85 52            STA    $0052
0384 A5 53            LDA    $0053
0386 69 00            ADCIM  $00
0388 85 53            STA    $0053
038A A5 52            LDA    $0052
038C 65 51            ADC    $0051  ADD X TO Y * 40.
038E 85 52            STA    $0052
0390 90 02            BCC    NOCHG
0392 E6 53            INC    $0053
0394 18        NOCHG  CLC
0395 A9 80            LDAIM  $80
0397 65 53            ADC    $0053
0399 85 53            STA    $0053
039B A0 10            LDYIM  $10    LOOK UP CHARACTER IN SCREEN
039D A2 00            LDXIM  $00    POSITION X+Y*40 IN TABLE
039F A1 52            LDAIX  $0052
03A1 88        CHARAC DEY
03A2 D9 BA 03         CMPY   TABLE
03A5 F0 09            BEQ    FOUND
03A7 C0 00            CPYIM  $00
03A9 D0 F6            BNE    CHARAC
03AB A6 B1            LDX    $00B1  IF NOT IN TABLE, CHECK $B1 FOR
03AD F0 01            BEQ    FOUND  USR ARGUMENT
03AF 60               RTS

03B0 98        FOUND  TYA
03B1 05 56            ORA    $0056  COMPUTE NEW CHARACTER
03B3 A8               TAY
03B4 B9 BA 03         LDAY   TABLE  STORE NEW CHARACTER ON SCREEN
03B7 81 52            STAIX  $0052
03B9 60               RTS

03BA 20        TABLE  =      $20    TABLE CONTAINS ALL SIXTEEN POSSIBLE
03BB 7E               =      $7E    PLOT CHARACTERS
03BC 7C               =      $7C
03BD E2               =      $E2
03BE 7B               =      $7B
03BF 61               =      $61
03C0 FF               =      $FF
03C1 EC               =      $EC
03C2 6C               =      $6C
03C3 7F               =      $7F
03C4 E1               =      $E1
03C5 FB               =      $FB
03C6 62               =      $62
03C7 FC               =      $FC
03C8 FE               =      $FE
03C9 A0               =      $A0
```

FIGURE 1

```
1   POKE  1,58:POKE 2,3
10  PRINT "(clr)"
20  FOR R=4 TO 16 STEP 4
30  P=38-R
40  Q=8+R
50  F=2*π/300
60  FOR I=0 TO 300
70  AN=F*I
80  X=INT(39.5+P*COS(AN))
90  Y=INT(24.5+Q*SIN(AN))
100 POKE 81,X:POKE 82,49-Y:A=USR(0)
110 NEXT I
120 NEXT R
130 GET G$:IF G$="" GOTO 130
```



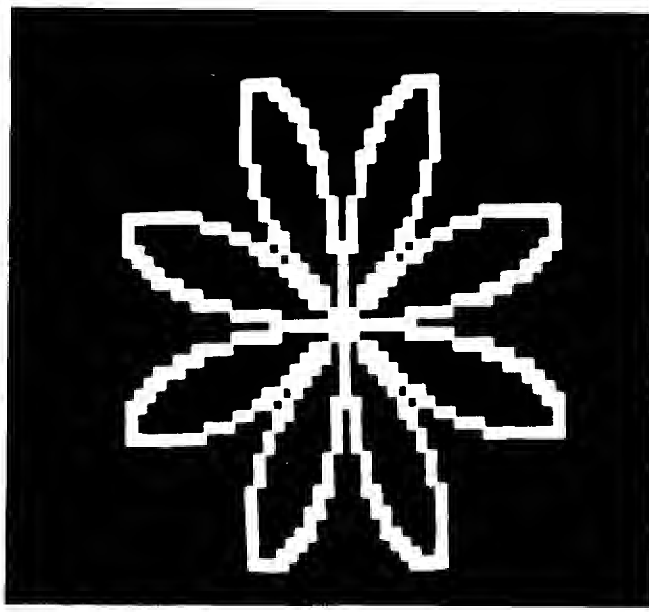


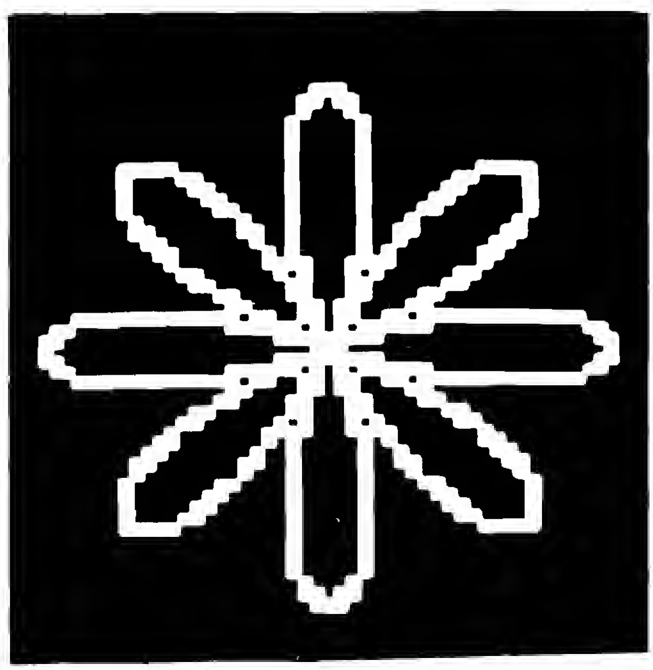
FIGURE 2A

```
1   POKE 1,58:POKE 2,3
10  PRINT "(clr)"
20  P=24:N=4
30  F=2*π/600
40  FOR I=0 TO 600
50  AN=I*F
55  R=P*SIN(N*AN)
60  X=INT(R*COS(AN)+39.5)
70  Y=INT(R*SIN(AN)+24.5)
80  POKE 81,X:POKE 82,49-Y:A=USR(0)
90  NEXT I
100 GET G$:IF G$="" GOTO 100
```

FIGURE 2B

(Changes to 2A only)

```
55 R=P*COS(N*AN)
```



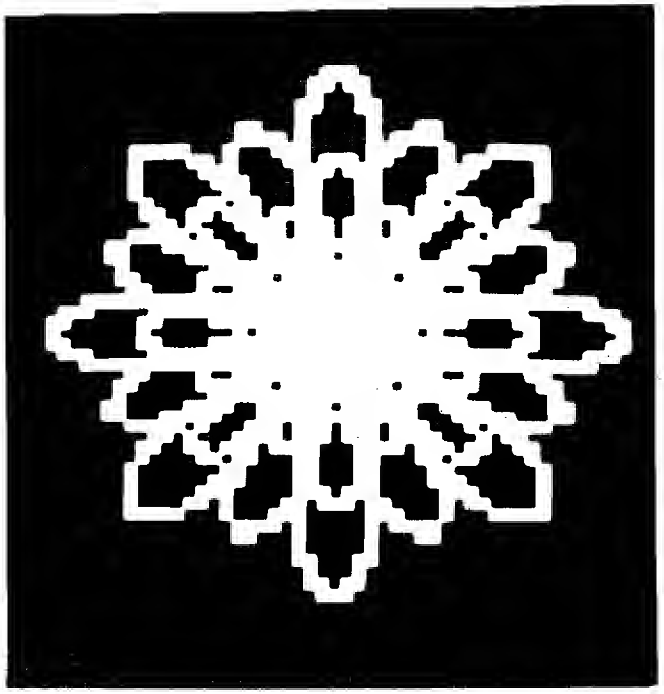


FIGURE 3

```
1    POKE 1,58:POKE 2,3
10   PRINT "(clr)"
20   P=9;Q=15/2
30   F=2* π/250
40   FOR I=0 TO 1250
50   AN=I*F
60   X=(P+Q)*COS(AN)+Q*COS((P+Q)*AN/Q)
70   Y=(P+Q)*SIN(AN)+Q*SIN((P+Q)*AN/Q)
80   X=INT(X+39.5):Y=INT(Y+24.5)
90   POKE 81,X:POKE 82,Y:A=USR(0)
100  NEXT I
110  GET G$: IF G$="" GOTO 110
```

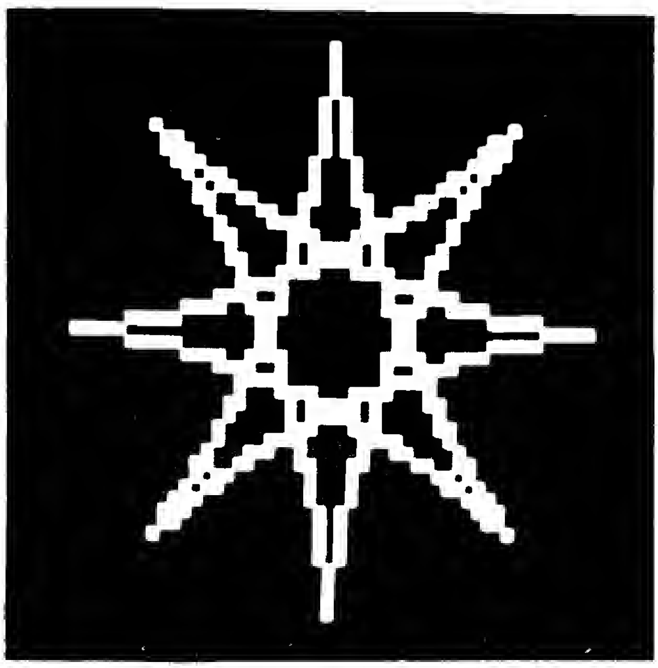


FIGURE 2C

(Changes to 2B only)

```
20   N=4
31   K1=1
32   FOR K2=0 TO 20 STEP 4
33   P=24-K2
34   K1=K1*-1
55   R=P*SIN(N*AN)
56   IF K1<0 THEN R=P*COS(N*AN)
```

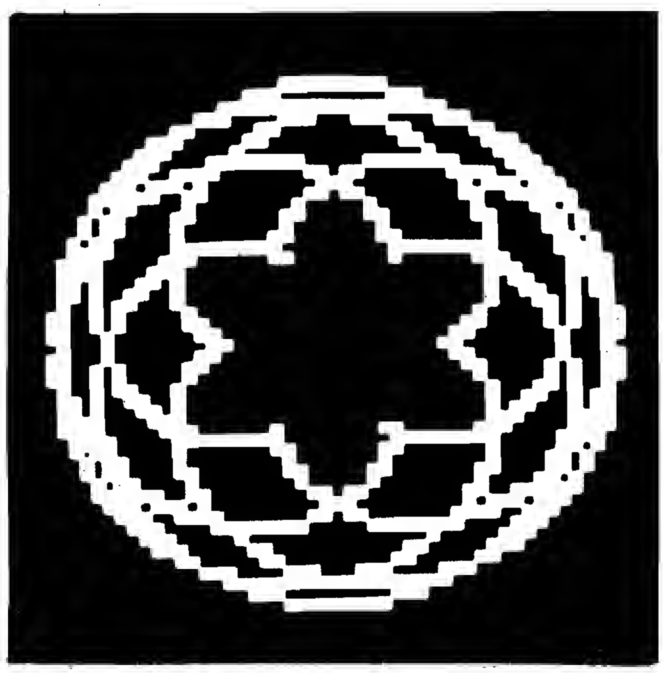


FIGURE 4

```
1    POKE 1,58:POKE 2,3
10   PRINT "(clr)"
20   P=24:Q=9
22   DT=300
24   F=2* π/DT
28   FOR I=1 TO 25
30   DL=P*I/Q-INT(P*I/Q)
32   IF DL<.00001 GOTO 36
34   NEXT I
36   PT=I*P/Q
38   PRINT "(home)";INT(PT+.5)
40   FOR J=0 TO I*DT
50   AN=J*F
60   X=(P-Q)*COS(AN)+Q*COS((P-Q)*AN/Q)
70   Y=(P-Q)*SIN(AN-Q*SIN((P-Q)*AN/Q)
80   X=INT(X+39.5):Y=INT(Y+24.5)
90   POKE 81,X:POKE 82,Y:A=USR(0)
100  NEXT J
110  GET G$:IF G$="" GOTO 110
```

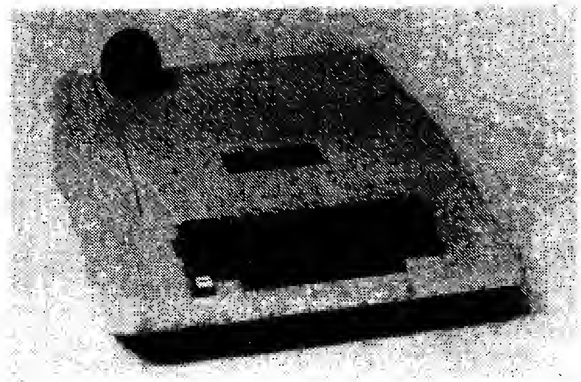
Computer Forum

· BUSINESS
· EDUCATIONAL
· PERSONAL

14052 EAST FIRESTONE BOULEVARD · SANTA FE SPRINGS, CALIFORNIA 90670

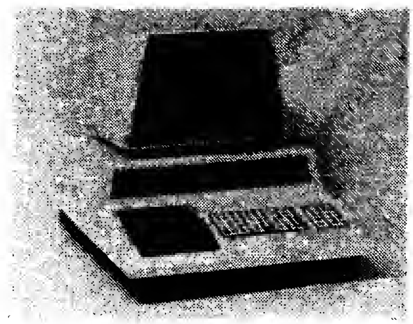
(213) 921-2111 (714) 739-0711

16K APPLE II $1195.00



- We are Apple Headquarters. Buy from us and let us help you get the most from your Apple.
- Free software with purchase of Apple. Choose up to $100.00 worth of software from over 100 selections in our Apple software catalog.
- Our unique Apple II software catalog gives a short critique on each program so that you know what you are buying before you buy.
- We also have numerous selections of software free with any purchase.
- We offer service contracts for all equipment we sell.
- We provide demonstrations and orientation services for schools and businesses anywhere in Los Angeles or Orange Counties, California. Call for information regarding this free service.

8K COMMODORE PET ... $795.00



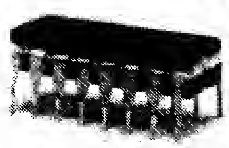
- FREE ...... $ 50.00 worth of software of your choice with each pet.
- Software .... Over 50 programs now available and still growing. Send for our software catalog.
- In Stock ..... Off the shelf deliveries
- Service ..... We have an on site service department.
- Users group now forming

BLANK CASSETTES

C-10 BLANK CASSETTES (W/O BOX) FOR MICRO COMPUTING.

$ 1.00 Ea. Other lengths will be available in the future.
$ 7.50 for 10 Call or write for quotes on larger quantities.
$32.50 for 50 Add 10% (minimum $2.00) for shipping and handling.

16K RAM FOR APPLE II



- 4116 Chips. 8 per set makes 16K RAM. Just any 4116 chips will not work in the Apple II. 350NS or slower will not work properly. 250NS chips are adequate. 200NS chips are preferable.
- 200NS chips. Tested and guaranteed .... $ 95.00 per set.
- Add $ 2.00 for shipping and handling.

QUANT.	ITEM	PRICE	
			ABOVE ITEMS ARE NORMAL STOCK ITEMS. DUE TO CIRCUMSTANCES BEYOND OUR CONTROL, WE ARE SOMETIMES TEMPORARILY OUT OF STOCK. PLEASE CHECK APPROPRIATE BOX BELOW.
			CERTIFIED CHECK, MONEY ORDER, VISA OR MASTER CHARGE ORDERS SHIPPED SAME DAY. NO COD. ALLOW 2 WEEKS FOR PERSONAL CHECK TO CLEAR.
	CALIFORNIA RESIDENTS, ADD 6% SALES TAX		PLEASE BACK ORDER IF OUT OF STOCK ☐ DO NOT BACK ORDER IF OUT OF STOCK ☐
	SHIPPING & HANDLING. ADD $10.00 FOR COMPUTER SYSTEM.		IF USING CREDIT CARD, CHECK BOX AND ENTER CARD NUMBER BELOW.
	TOTAL		MASTER CHARGE ☐ VISA ☐

USING TINY BASIC TO DEBUG MACHINE LANGUAGE PROGRAMS

Jim Zuber
20224 Cohasset No. 16
Canoga Park, CA 91306

I just got Tiny BASIC up and running on my KIM-1 and have found it to be a valuable enhancement to writing machine (or assembly) language programs. The Tiny BASIC USR function allows us to access machine language subroutines from within a BASIC program. You can pass parameters to and from the BASIC program and the machine language subroutine. If you can make an entire machine language program appear as a subroutine to Tiny BASIC (add a RTS call in the appropriate place) then Tiny BASIC can access your entire program with the USR function. A natural application of this capability is a debugging program written in Tiny BASIC that can completely test a machine language subroutine without ever leaving the Tiny BASIC program. The only limitation to this is that your machine language program cannot reside in the memory area used by Tiny BASIC and you must not use the same zero page locations as Tiny BASIC. My program (see listing #1) will print out the data in 4 memory locations when a predefined set of conditions exist in the machine language subroutine. There are 7 user selectable functions in the command mode:

(0) DEFINE SUBROUTINE ADDRESS - This is the starting address of the machine language subroutine you want to test.

(1) DEFINE PRINT ADDRESS - This allows you to define the conditions that must exist before data is printed out in the run mode. There are 3 options:
(A) Print every loop through the subroutine.
(B) Print at a predefined loop interval (use a decimal number).
(C) Conditional print - Program will only print out when data in a predefined address matches the value specified.
The print mode is initialized at "Print every loop" at the start of the program.

(3) PRESET DATA - This allows the user to place data in any memory location.

(4) PRINT LIMIT - This number limits the number of times the program prints out the 4 addresses when in the run mode. This is initialized at 10 in the beginning of the program. Use a decimal value for the print limit.

(5) RUN PROGRAM - This starts the Tiny BASIC program looping through and printing out data from the machine language subroutine.

(6) EXIT PROGRAM - Returns you to Tiny BASIC monitor.

(7) See COMMAND OPTIONS.

All command options except 6 return to the command mode after execution. If your version of Tiny BASIC does not start at hex 2000 you must change line 50 to the decimal equivalent of your Tiny BASIC starting address. All address and data questions should be answered in hex with a comma between each digit. Example: Address 0,2,0,0 or Data A,2. The program will print a marker every 50 loops through the machine language subroutine. This can be changed to suit your preference by modifying line 295. The following example should clarify the functions and use of the Debug Program. Listing #2 is a subroutine from a Biorhythm Program that I wrote. The subroutine increments 3 memory locations that correspond to the physical, emotional, and intellectual biorhythm cycle days. Each memory location should be reset to day one at the appropriate point.

Location 0001 = 1 to 23 days (physical)
Location 0002 = 1 to 28 days (emotional)
Location 0003 = 1 to 33 days (intellectual)

Two days in each cycle are considered critical. They are.

Physical: Day 1 and day 12
Emotional: Day 1 and day 15
Intellectual: Day 1 and day 17

Location 0004 is incremented for each cycle that is critical on a particular pass through the subroutine. An 02 in this location would indicate a double critical day on that particular pass through the subroutine. As a subroutine of this type would take several hours to test using conventional methods due to the large number of variables, the following sample runs from the Tiny BASIC machine Debug Program will show how complete testing of a subroutine can be done in a few minutes. (See samples #1 through #4).

Sample #1 The starting address is set to 0200 and the print addresses are defined as 0001 through 0004. The printout shows that locations 0001 - 0003 are incrementing as they should.

Sample #2 Memory locations 0001 - 0003 are preset to day 20 and the print limit is set to 15. The printout shows that the cycles are resetting to day 1 at the appropriate time. (days 23, 28, and 33).

Sample #3 The print limit is set to 4 and the print mode is set to every 23rd loop in order to check the consistency of the subroutine. The printout shows that location 0001 is staying the same as it should. (location 0001 is the 23 day physical cycle). Note the marker at 50 loops.

Sample #4 The print mode is set to the conditional mode in order to print only when location 0004 is equal to 02 (double citical day). The printout shows the subroutine is working properly.

I would like to thank Tom Pittman (author of Tiny BASIC) whose programming tricks in the Tiny BASIC User Manual made this program possible. I hope the Machine Debug Program can take the sweat out of testing your subroutines!

----------LISTING #1----------

```
10 REM TINY BASIC MACHINE DEBUG PROGRAM
11 REM BY JIM ZUBER---SEPT 29,1978
15 A=-10
20 B=-11
25 C=-12
30 D=-13
35 E=-14
40 F=-15
47 REM TINY START ADDRESS(DEC)
50 S=8192
54 REM PRESET PRINT LIMIT
55 G=10
59 REM PRESET PRINT MODE
60 H=1
65 REM ANSWER ALL ADDRESS AND DATA
70 REM QUESTIONS WITH A HEX NUMBER
75 REM THAT HAS EACH DIGIT SEPERATED
80 REM BY A COMMA.
85 REM AT A MINIMUM SET SUB ADDRESS
90 REM AND 4 DATA ADDRESSES.
100 PR "COMMAND MODE----SELECT ONE"
102 PR "   0. DEFINE SUBROUTINE ADDRESS"
103 PR "   1. DEFINE PRINT ADDRESSES"
106 PR "   2. DEFINE PRINT MODE"
109 PR "   3. PRESET DATA"
112 PR "   4. PRINT LIMIT"
115 PR "   5. RUN PROGRAM"
118 PR "   6. EXIT PROGRAM"
119 PR "   7. SEE COMMAND OPTIONS"
121 INPUT L
123 IF L=0 GOTO 275
124 IF L=1 GOTO 148
127 IF L=2 GOTO 187
130 IF L=3 GOTO 239
133 IF L=4 GOTO 266
136 IF L=5 GOTO 281
139 IF L=6 GOTO 1000
141 IF L=7 GOTO 102
142 PR "---COMMAND MODE---"
145 GOTO 121
147 REM INPUT 4 ADDRESSES
148 PR "POSITION #1--";
151 GOSUB 800
154 T=N
157 PR "POSITION #2--";
160 GOSUB 800
163 U=N
166 PR "POSITION #3--";
169 GOSUB 800
172 V=N
175 PR "POSITION #4--";
178 GOSUB 800
181 W=N
184 GOTO 142
186 REM DEFINE PRINT MODE
187 PR "PRINT MODE---SELECT ONE"
190 PR "   1. ALL LOOPS"
193 PR "   2. DEFINE NUMBER OF LOOPS"
196 PR "   3. CONDITIONAL PRINT"
199 INPUT H
202 IF H=1 GOTO 142
205 IF H=2 GOTO 215
208 IF H=3 GOTO 224
211 GOTO 187
215 PR "INPUT LOOP INCREMENT"
218 INPUT K
221 GOTO 142
223 REM CONDITIONAL PRINT MODE
224 GOSUB 800
227 I=N
230 GOSUB 850
233 J=N
236 GOTO 142
238 REM PRESET DATA
239 GOSUB 800
242 P=N
245 GOSUB 850
248 Q=N
251 Y=USR(S+24,P,Q)
254 PR "ANY MORE TO PRESET?(1=N 2=Y)"
257 INPUT Y
260 IF Y=2 GOTO 239
263 GOTO 142
265 REM SET PRINT LIMIT
266 PR "INPUT PRINT LIMIT"
269 INPUT G
272 GOTO 142
274 REM DEFINE SUB START ADDRESS
275 GOSUB 800
276 O=N
278 GOTO 142
280 REM RUN PROGRAM
281 PR
284 PR "LOOP","-1-","-2-","-3-","-4-"
287 PR
289 X=0
290 P=0
293 P=P+1
294 Y=USR(O)
295 IF P=P/50*50 THEN PR P
296 IF H=1 GOTO 314
299 IF H=2 GOTO 323
301 REM CONDITIONAL PRINT
302 Y=USR(S+20,I)
305 IF J=Y GOSUB 500
308 IF X=G GOTO 142
```

----------LISTING #1 CONTINUED----------

```
311 GOTO 293
313 REM PRINT ALL LOOPS
314 GOSUB 500
317 IF X=G GOTO 142
320 GOTO 293
322 REM DEFINED NUMBER OF LOOPS
323 IF P=P/K*K GOSUB 500
325 IF X=G GOTO 142
327 GOTO 293
349 REM SUB TO PRINT 2 HEX DIGITS
350 M=Z/16
355 Z=Z-M*16
360 GOSUB 400+M+M
370 GOSUB 400+Z+Z
375 RETURN
400 PR 0;
401 RETURN
402 PR 1;
403 RETURN
404 PR 2;
405 RETURN
406 PR 3;
407 RETURN
408 PR 4;
409 RETURN
410 PR 5;
411 RETURN
412 PR 6;
413 RETURN
414 PR 7;
415 RETURN
416 PR 8;
417 RETURN
418 PR 9;
419 RETURN
420 PR "A";
421 RETURN
422 PR "B";
423 RETURN
424 PR "C";
425 RETURN
426 PR "D";
427 RETURN
428 PR "E";
429 RETURN
430 PR "F";
431 RETURN
499 REM SUB TO PRINT 4 ADDRESSES
500 PR P,
505 Z=USR(S+20,T)
510 GOSUB 350
515 PR "      ";
520 Z=USR(S+20,U)
525 GOSUB 350
530 PR "      ";
535 Z=USR(S+20,V)
540 GOSUB 350
545 PR "      ";
550 Z=USR(S+20,W)
555 GOSUB 350
560 PR
565 X=X+1
570 RETURN
800 REM SUB ADDRESS(HEX TO DEC)
802 N=0
805 X=1
807 PR "INPUT ADDRESS"
810 INPUT R
815 GOSUB 900
820 IF X=4 RETURN
825 X=X+1
830 GOTO 810
850 REM SUB DATA(HEX TO DEC)
852 N=0
855 X=1
857 PR "INPUT DATA"
860 INPUT R
865 GOSUB 900
870 IF X=2 RETURN
875 X=X+1
880 GOTO 860
900 REM HEX TO DECIMAL SUB
905 IF R>999 THEN N=N*16
910 IF R>99 THEN N=N*16
915 IF R>9 THEN N=N*16
920 IF R>0 GOTO 990
925 IF R<0 THEN R=-R
930 N=N*16+R
935 RETURN
990 R=R+R/1000*1536+R/100*96+R/10*6
995 GOTO 925
1000 END
```

----------LISTING #2----------

```
                            .OR 0200       START ADDRESS
                      PHY.  .DL 0001       LABELS
                      EMT.  .DL 0002
                      INT.  .DL 0003
:                     CRIT  .DL 0004
0200  A9 00                 LDA 00         START
0202  85 04                 STA *CRIT
0204  F8              STAR  SED
0205  18                    CLC
0206  A5 01                 LDA *PHY.      INCREMENT PHY
0208  C9 23                 CMP 23
020A  F0 07                 BEQ SET1
020C  69 01                 ADC 01
020E  85 01                 STA *PHY.
0210  4C 17 02              JMP EMOT
0213  A9 01           SET1  LDA 01
0215  85 01                 STA *PHY.
0217  A5 02           EMOT  LDA *EMT.      INCREMENT EMT
0219  C9 28                 CMP 28
021B  F0 07                 BEQ SET2
021D  69 01                 ADC 01
021F  85 02                 STA *EMT.
0221  4C 28 02              JMP INTL
0224  A9 01           SET2  LDA 01
0226  85 02                 STA *EMT.
0228  A5 03           INTL  LDA *INT.      INCREMENT INT
022A  C9 33                 CMP 33
022C  F0 07                 BEQ SET3
022E  69 01                 ADC 01
0230  85 03                 STA *INT.
0232  4C 39 02              JMP PCRT
0235  A9 01           SET3  LDA 01
0237  85 03                 STA *INT.
0239  A5 01           PCRT  LDA *PHY.      PHY CRITICAL?
023B  C9 01                 CMP 01
023D  F0 19                 BEQ LOP1
023F  C9 12                 CMP 12
0241  F0 15                 BEQ LOP1
0243  A5 02           ECRT  LDA *EMT.      EMT CRITICAL?
0245  C9 01                 CMP 01
0247  F0 14                 BEQ LOP2
0249  C9 15                 CMP 15
024B  F0 10                 BEQ LOP2
024D  A5 03           ICRT  LDA *INT.      INT CRITICAL?
024F  C9 01                 CMP 01
0251  F0 0F                 BEQ LOP3
0253  C9 17                 CMP 17
0255  F0 0B                 BEQ LOP3
0257  60              EXIT  RTS
0258  E6 04           LOP1  INC *CRIT      INCREMENT CRIT
025A  4C 43 02              JMP ECRT
025D  E6 04           LOP2  INC *CRIT
025F  4C 4D 02              JMP ICRT
0262  E6 04           LOP3  INC *CRIT
0264  4C 57 02              JMP EXIT
                      END.  .EN
```

SYMBOL	TABLE
PHY.	0001
EMT.	0002
INT.	0003
CRIT	0004
STAR	0204
SET1	0213
EMOT	0217
SET2	0224
INTL	0228
SET3	0235
PCRT	0239
ECRT	0243
ICRT	024D
EXIT	0257
LOP1	0258
LOP2	025D
LOP3	0262
END.	0267

```
----------SAMPLE #1----------

:RUN

COMMAND MODE----SELECT ONE
   0. DEFINE SUBROUTINE ADDRESS
   1. DEFINE PRINT ADDRESSES
   2. DEFINE PRINT MODE
   3. PRESET DATA
   4. PRINT LIMIT
   5. RUN PROGRAM
   6. EXIT PROGRAM
   7. SEE COMMAND OPTIONS
? 0

INPUT ADDRESS
? 0,2,0,0

---COMMAND MODE---
? 1

POSITION #1--INPUT ADDRESS
? 0,0,0,1

POSITION #2--INPUT ADDRESS
? 0,0,0,2

POSITION #3--INPUT ADDRESS
? 0,0,0,3

POSITION #4--INPUT ADDRESS
? 0,0,0,4

---COMMAND MODE---
? 5

LOOP    -1-    -2-    -3-    -4-

1       02     02     12     00
2       03     03     13     00
3       04     04     14     00
4       05     05     15     00
5       06     06     16     00
6       07     07     17     01
7       08     08     18     00
8       09     09     19     00
9       10     10     20     00
10      11     11     21     00
```

```
----------SAMPLE #2----------
---COMMAND MODE---
? 3

INPUT ADDRESS
? 0,0,0,1

INPUT DATA
? 2,0

ANY MORE TO PRESET?(1=N 2=Y)
? 2

INPUT ADDRESS
? 0,0,0,2

INPUT DATA
? 2,0

ANY MORE TO PRESET?(1=N 2=Y)
? 2

INPUT ADDRESS
? 0,0,0,3

INPUT DATA
? 2,0

ANY MORE TO PRESET?(1=N 2=Y)
? 1

---COMMAND MODE---
? 4

INPUT PRINT LIMIT
? 15

---COMMAND MODE---
? 5

LOOP    -1-    -2-    -3-    -4-
1       21     21     21     00
2       22     22     22     00
3       23     23     23     00
4       01     24     24     01
5       02     25     25     00
6       03     26     26     00
7       04     27     27     00
8       05     28     28     00
9       06     01     29     01
10      07     02     30     00
11      08     03     31     00
12      09     04     32     00
13      10     05     33     00
14      11     06     01     01
15      12     07     02     01
```

```
----------SAMPLE #3----------
---COMMAND MODE---
? 4

INPUT PRINT LIMIT
? 4

---COMMAND MODE---
? 2

PRINT MODE---SELECT ONE
   1. ALL LOOPS
   2. DEFINE NUMBER OF LOOPS
   3. CONDITIONAL PRINT
? 2

INPUT LOOP INCREMENT
? 23

---COMMAND MODE---
? 5


LOOP    -1-     -2-     -3-     -4-

23      12      02      25      01
46      12      25      15      01
50
69      12      20      05      01
92      12      15      28      02
```

```
----------SAMPLE #4----------
---COMMAND MODE---
? 2

PRINT MODE---SELECT ONE
   1. ALL LOOPS
   2. DEFINE NUMBER OF LOOPS
   3. CONDITIONAL PRINT
? 3

INPUT ADDRESS
? 0,0,0,4

INPUT DATA
? 0,2

---COMMAND MODE---
? 5


LOOP    -1-     -2-     -3-     -4-

50
100
138     12      13      01      02
150
154     05      01      17      02
196     01      15      26      02
200
250
253     12      16      17      02
---COMMAND MODE---
? 6
```

ASK THE DOCTOR — PART II
AN ASK EPROM PROGRAMMER

Robert M. Tripp, Ph. D.
The COMPUTERIST, Inc.
P.O. Box 3
So. Chelmsford, MA 01824

One of the most frequently asked questions about the **ASK** (AIM/SYM/KIM) family of microcomputers is: "Can a program that was written for one of the micros run on either of the others?" The answer is normally no. While the three micros share a lot - common expansion bus, similar application connector, KIM tape format ... they do have minor differences in their use of page zero and page one, some greater differences in their memory and I/O allocations, and large differences in their monitor subroutines. Therefore, in general, the answer to the question is: "No, a program written to run on one will not run on the others without modification." This answer may lead the creative programmer to wonder what it would take to write programs which would run on all three machines, without requiring customization for each. What problems would be encountered? What techniques could be used to reduce the problems? What about ...?

I faced the three-machine problem for a practical reason. the MEMORY PLUS™ board that my company makes is hardware compatible on the three systems. Part of the package is a cassette tape with a Memory Test program and an EPROM Programming program. It would be awkward to have to provide three sets of programs on the tape and expensive to have to print up three different sets of program listings. Would it be feasible to write a single program? The answer turned out to be: "Yes". The program for the EPROM Programmer is presented here in its entirity.

There are two major types of compatibility problems. The first is that the three monitors each have a different set of support subroutines. Sometimes they may have identical subroutines, but usually the subroutines are not identical, and often are not even close! In this particular program, this was not a problem since the program did not use any monitor subroutines. The second major problem is that various important locations in memory or in memory mapped I/O are different on the three systems. Examples are the re-entry address for returning to the monitor at the end of the program, the location of the interrupt vector, and the address of the peripheral I/O port. In this program all three of these address problems were encountered. The solution for the addressing problem is fairly simple and will handle all three addressing problems - if you understand the **Indirect Indexed** mode of addressing on the 6502. If you are totally unfamiliar with this addressing mode, you should consult your programming manual at this point and find out about it. If you are familiar with it, then this review may be useful.

The **Indirect Indexed** addressing mode on the 6502 works by having a base pointer in a pair of page zero locations which is used to point to some other location in memory. The contents of the page zero locations are combined with current contents of the Y register to form the final address for an instruction. The assembler form of the instruction is LDA (POINT), Y in the standard MOS Technology syntax or LDAIY POINT in the MICRO-ADE syntax which is generally used in MICRO. In either case, what results is a form of addressing in which the page zero pointer forms the base address and the contents of the Y register allow this address to be modified within a range of 00 to FF. If the pointer value was 2800, then the effective range of the indirect indexed instruction would be 2800 (with Y = 00) to 28FF (with Y = FF). The page zero pointer is set up in two consecutive bytes, with the low byte of the address first followed by the high byte of the address. In our example, if POINT was the page zero address 0006, then location 0006 would contain 00 (the low byte of the indirect address) and 0007 would contain 28 (the high byte of the indirect address). Since the only problem we have to solve for the EPROM Programmer is one of different addresses for the three systems, the problem reduces to three steps:

1. Determine which system we are runnng on: AIM, SYM or KIM.
2. Set up appropriate indirect address pointers.
3. Access the variable addresses via the indirect address pointers using the Indirect Indexed addressing mode.

Now Let's examine the program in a little detail to see how it actually accomplishes all of this.

The Program

The program is assembled to run entirely on page zero. It uses a 6522 VIA chip which is located on the MEMORY PLUS board for a lot of its I/O and timing. The registers within the VIA that are used are listed under VIA REGISTER OFFSETS. These offsets will be used within the program to load the Y register prior to making an Indirect Indexed instruction call so that the appropriate VIA internal register will be accessed. The first six locations in page zero are used by the program for parameters to control where the data to be placed into the EPROM starts in memory, ends in memory, and where it is to be placed in the EPROM. This information is filled in by the operator before running the program. Location **"VIA"** is an indirect pointer to the MEMORY PLUS VIA chip. This normally will be at location 6200 and could have been addressed directly by the program. But, since it could be in another address, it was decided to handle it through the Indirect Indexed mode. The **"JMPMON"** location contains the Op-code for a JMP. This is used in conjunction with the contents of the next two bytes, **"MONTOR"**, to re-enter the system monitor at the end of the program or when an error is encountered. The actual monitor re-entry address value is filled in by the program. It appears as 0000 in the listing, but will be altered early in the program as we shall see below. The **"INTVEC"** is an indirect pointer to the IRQ interrupt vector which is used as part of the timing service of the program. This will be properly filled in at the beginning of the program from a table. **"PBDD"** and **"PBD"** are pointers to the Port B Data Direction and Port B Data registers. These will also be filled in from a table at the start of the program and will be used in Indirect Indexed instructions.

The program begins execution at location 0011, after the user has used his monitor to fill in the appropriate values in the parameters in locations 0000 to 0005. The first three instructions clear all of the status bits by pushing a 00 onto the stack from A and popping it into the status register.

Locations 0015 through 0027 determine which microcomputer the program is running on by testing the contents of a ROM location. The contents of location FFFD is specific to each machine. This is the high order byte of the Reset Interrupt Vector. For the SYM this will be an 8B; for the AIM an E0, and for the KIM a 1C. The X register is loaded with a value which is the start of a table of values which will be moved into locations 0009 through 0010 to fill in the **MONTOR, INTVEC, PBDD,** and **PBD** pointers discussed above. The instruction at 0028 is unique to the SYM and is required to permit the program to access some of the SYM's protected memory locations. It is not executed by the program for KIM or AIM.

Locations 002B through 0035 move the appropriate table from its original location at the end of the program into the working indirect area. The AIM table starts at 00D0; the KIM table at 00D8; the SYM at 00E0.

By the time we reach **ENTER** at location 0036, two important things have been done. First, we have determined which machine we are running on. Second, using this information, we have set up our indirect pointers which will be used by the remainder of the program to address the machine specific addresses. At **ENTER** we again set the status bits to zero. This is done so that a user with a different computer could still use this program. He would do this by manually setting up the pointers in 0009 through 0010 and then starting at 0036 - **ENTER.**

Locations 003A through 0044 fill in the system interrupt vector to point to the interrupt servicing routine of the program which starts at 00C5. This is a good place to examine the workings of the Indirect Indexed addressing. The Y register is set to 00. The A register is loaded with the low byte of the interrupt service routine address. This value will be C5 since the routine starts at 00C5. This is then stored in the system interrupt vector which is addressed by adding the contents of Y (00) to the address contained in **INTVEC.** For the AIM **INTVEC** will have been set to A400; for the KIM **INTVEC** will be 17FE; for the SYM A67E. So the effective address will be A400 for the AIM (A400 + 00= A400), 17FE for the KIM and A67E for the SYM. The A register is then loaded with the high byte of the interrupt service routine address, 00 since the routine is in page zero. The Y register is incremented so that it now contains a 01. When A is now stored with Indirect Indexed mode through **INTVEC,** it goes into A401 on the AIM (A400 + 01 = A401), 17FF on the KIM and A67F on the SYM. If you are not clear at this point as to how this works, then **STOP.** The rest of this article will make no sense until you understand the basics of the Indirect Indexed mode. Re-read the article to this point, consult your manual, ask a friend.

Using the same techniques of setting Y to an offset value, loading A with the value to use, and storing in the Indirect Indexed mode, the VIA is initialized.

The instructions from 005D through 0078 set up the VIA for output. One additional trick is used here. While we normally think of the Y register in connection with the Indirect Indexed mode of addressing, the X register can also be used for this mode of addressing - but only under one special condition. That condition is when the index value is 00. In this condition, the Indirect Indexed mode and the Indexed Indirect mode both collapse to the simple Indirect mode. There are several places in which we take advantage of this fact so that the X register can be set to zero once and used several times for addressing. This section of code now gets the data from the indirect pointers that the operator set into locations 0000 through 0005 and outputs the data to the EPROM Programmer.

Locations 0079 through 008A first set a timer in the VIA going for the 50 millisecond period which is required to program one location on the EPROM. Then the Peripheral Control Register on the VIA is set to enable the programming pulse to the EPROM. Again, Indirect Indexed addressing is used so that the VIA does not have to be at 6200. If it is in any other address, the operator simply sets the pointer at VIA (0006, 0007) before starting the program. Everything else is automatic.

Locations 008B to 008E form a loop which waits until an interrupt has occurred and been serviced. If you look down at the interrupt routine starting at 00C5 you will see that Y is changed so that it is no longer equal to 0C. At this point the **WAIT** test will fail and the program will move on to **VERIFY.**

Locations 008F through 00C4 perform a series of tests and pointer updates. When the program reaches the end of the data, or if it detects an error, it makes a **JSR** to **JMPMON**. **JMPMON** then jumps to a re-entry point for the appropriate monitor as set up from the table at the beginning of the program. The reason for making the JSR is to save the address of where we are coming from to be displayed by the monitor as an indication of why we exited: successful completion or one of the three errors. The **JMPMON** permits us to go to the correct monitor. While it would have been possible to have the initialization code change each of the four JSR's to JSR directly to the appropriate monitor, this obviously would have entailed more code and would not have any benefit.

The re-entry to the monitor is the only place where this code makes use of the system monitor, and wouldn't you know it - each monitor handles the re-entry slightly differently. They each display an address which is related to the JSR from which it came, but each one displays a slightly different address. On the successful completion return which is at 00B7, the AIM displays 00B8, the KIM displays 00B9, and the SYM displays 00BA. It would have been possible to write some additional code to take care of the address before returning to the monitor, but this did not seem to be a serious enough problem to warrant the effort. But it does point out the problems one can encounter in using the "similar-but-different" monitor subroutines.

Locations 00C5 through 00CF are the interrupt service. When the interrupt occurs, it is vectored here due to the setup that took place earlier in the program. The VIA is changed from programming mode to verify mode and the interrupt is cleared. In the process the Y register is changed so that the **WAIT** test will permit the program to recognize that an interrupt has occurred and to continue.

The **ATABLE, KTABLE** and **STABLE** are the pointer values for the AIM, KIM and SYM respectively. At the start of the program they are moved into a standard set of locations starting at 0009 (**MONTOR**).

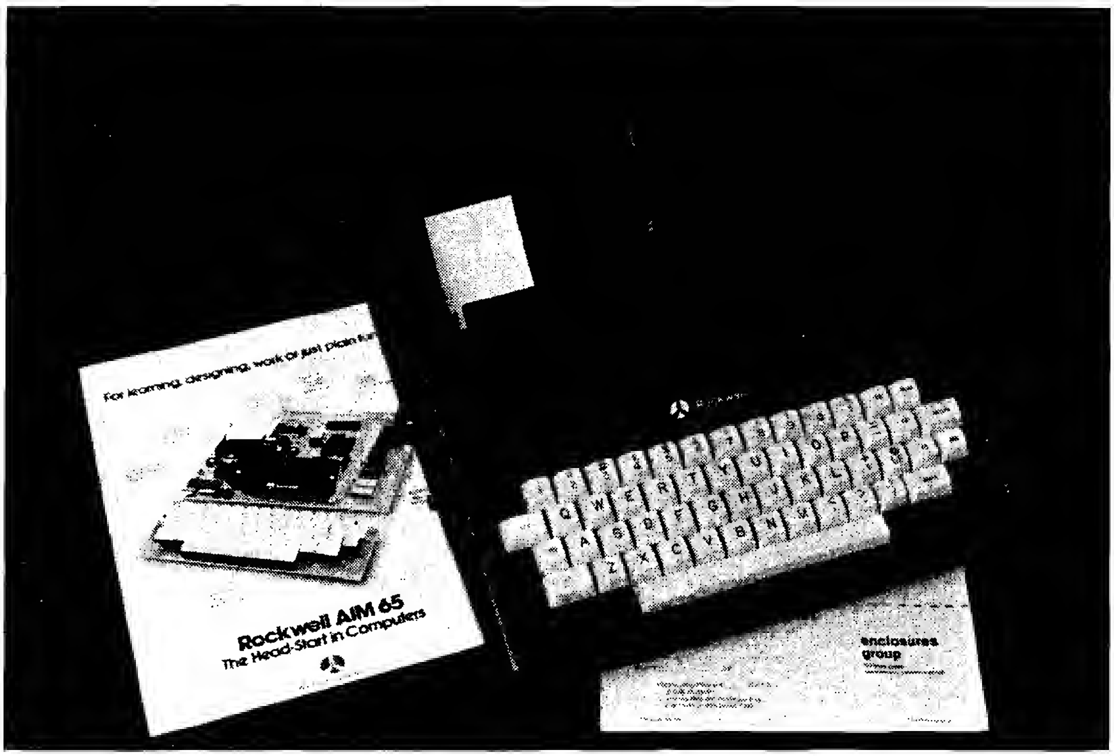
Next month, ASK the Doctor will present the hardware that is required to build an EPROM Programmer based on this program.

```
                  PROM PROGRAMMER  10 FEBRUARY 1979

                  PROM    ORG    $0000

                  ACCESS *       $8B86  SYM-1 ACCESS ENTRY

                  VIA REGISTER OFFSETS

                  ORB     *      $0000  OUTPUT REGISTER B
                  ORA     *      $0001  OUTPUT REGISTER A
                  DDRB    *      $0002  DATA DIRECTION REGISTER B
                  DDRA    *      $0003  DATA DIRECTION REGISTER A
                  TTWOL   *      $0008  TIMER TWO LOW
                  TTWOH   *      $0009  TIMER TWO HIGH
                  PCR     *      $000C  PERIPHERAL CONTROL REGISTER
                  IFR     *      $000D  INTERRUPT FLAG REGISTER
                  IER     *      $000E  INTERRUPT ENABLE REGISTER

0000 00           SAL     =      $00    STARTING ADDRESS LOW
0001 00           SAH     =      $00    STARTING ADDRESS HIGH
0002 00           PRMLOW  =      $00    EPROM LOW ADDRESS
0003 00           PRMHGH  =      $00    EPROM HIGH ADDRESS
0004 00           EAL     =      $00    END ADDRESS LOW
0005 00           EAH     =      $00    END ADDRESS HIGH
0006 00           VIA     =      $00    POINTER TO VIA
0007 62                   =      $62    NORMALLY AT 6200
0008 4C           JMPMON  =      $4C    JUMP TO MONITOR
0009 00           MONTOR  =      $00    POINTER TO SYSTEM MONITOR
000A 00                   =      $00    FOR RETURN FROM PROGRAMMER
000B 00           INTVEC  =      $00    POINTER TO INTERRUPT VECTOR
000C 00                   =      $00
000D 00           PBDD    =      $00    PORT B DATA DIRECTION
000E 00                   =      $00
000F 00           PBD     =      $00    PORT B DATA
0010 00                   =      $00

0011 A9 00        BEGIN   LDAIM  $00    CLEAR ALL STATUS FLAGS
0013 48                   PHA
0014 28                   PLP
0015 A2 E0                LDXIM  STABLE ASSUME SYM
0017 AD FD FF             LDA    $FFFD  TEST HIGH BYTE OF INTERRUPT VECTOR
001A C9 8B                CMPIM  $8B    = 8B FOR SYM-1
001C F0 0A                BEQ    SYM
001E A2 D0                LDXIM  ATABLE ASSUME AIM 65
0020 C9 E0                CMPIM  $E0    = E0 FOR AIM 65
0022 F0 07                BEQ    MOVE   IT IS THE AIM
0024 A2 D8        KIM     LDXIM  KTABLE ASSUME KIM
0026 D0 03                BNE    MOVE
0028 20 86 8B     SYM     JSR    ACCESS SYM REQUIRES ACCESS

002B 86 30        MOVE    STXZ   TABLE  +01 SETUP POINTER
002D A2 07                LDXIM  $07    MOVE 8 BYTES
002F B5 00        TABLE   LDAX   $00    REPLACED BY TABLE
0031 95 09                STAX   MONTOR MOVE TO MONTOR TABLE
0033 CA                   DEX
0034 10 F9                BPL    TABLE  MOVE UNTIL X = FF
```

```
0036 A9 00      ENTER  LDAIM $00    CLEAR ALL STATUS FLAGS
0038 48                PHA
0039 28                PLP
003A A0 00             LDYIM $00    ENTRY IF TABLE PRESET
003C A9 C5             LDAIM INTRPT GET INTERRUPT POINTER
003E 91 0B             STAIY INTVEC SETUP IN TABLE
0040 A9 00             LDAIM INTRPT /
0042 C8                INY          BUMP POINTER
0043 91 0B             STAIY INTVEC
0045 A9 EC             LDAIM $EC    SETUP VIA VALUES
0047 A0 0C             LDYIM PCR
0049 91 06             STAIY VIA
004B A0 0E             LDYIM IER    DISABLE ALL INTERRUPTS
004D A9 7F             LDAIM $7F
004F 91 06             STAIY VIA
0051 A0 0D             LDYIM IFR
0053 A9 FF             LDAIM $FF    CLEAR INTERRUPT PENDING
0055 91 06             STAIY VIA
0057 A0 0E             LDYIM IER
0059 A9 A0             LDAIM $A0    ENABLE TIMER TWO
005B 91 06             STAIY VIA

005D A2 00      NEXT   LDXIM $00    INIT X REGISTER
005F A9 FF             LDAIM $FF    SET DATA DIRECTION
0061 A0 02             LDYIM DDRB
0063 91 06             STAIY VIA
0065 A0 03             LDYIM DDRA
0067 91 06             STAIY VIA
0069 81 0D             STAIX PBDD
006B A5 02             LDA   PRMLOW OUTPUT NEXT ADDRESS
006D 81 06             STAIX VIA    LOW 8 BITS
006F A5 03             LDA   PRMHGH
0071 81 0F             STAIX PBD    BITS 8, 9, 10
0073 A1 00             LDAIX SAL    GET DATA BYTE
0075 A0 01             LDYIM ORA
0077 91 06             STAIY VIA    OUTPUT VIA ORA

0079 A9 50      TIMER  LDAIM $50    SETUP 50 MILLISECOND TIMER
007B A0 08             LDYIM T2WCL
007D 91 06             STAIY VIA    OUTPUT TO TIMER TWO LOW
007F A9 C3             LDAIM $C3    HIGH BYTE OF TIMER
0081 A0 09             LDYIM T2WCH
0083 91 06             STAIY VIA    OUTPUT TO TIMER TWO HIGH
0085 A9 CE             LDAIM $CE    PROGRAM HIGH, PROGRAM MODE
0087 A0 0C             LDYIM PCR
0089 91 06             STAIY VIA

008B C0 0C      WAIT   CPYIM PCR    TEST FOR INTERRUPT SERVICED
008D F0 FC             BEQ   WAIT   ELSE, WAIT FOR IT

008F A9 00      VERIFY LDAIM $00    VERIFY PROGRAMMING
0091 A0 03             LDYIM DDRA   SET ORA FOR INPUT
0093 91 06             STAIY VIA
0095 A0 01             LDYIM ORA    SETUP POINTER
0097 B1 06             LDAIY VIA
```

```
0099 C1 00          CMPIX SAL    COMPARE ORIGINAL DATA
009B F0 03          BEQ   OKAY   GOOD IF MATCH
009D 20 08 00       JSR   JMPMON EXIT ON ERROR

00A0 E6 00    OKAY  INC   SAL    BUMP DATA POINTER
00A2 D0 07          BNE   TEST   BRANCH IF NOT ZERO
00A4 E6 01          INC   SAH    BUMP HIGH DATA POINTER
00A6 D0 03          BNE   TEST   BRANCH IF NOT ZERO
00A8 20 08 00       JSR   JMPMON EXIT ON ERROR

00AB A5 05    TEST  LDA   EAH    TEST ALL DONE
00AD C5 01          CMP   SAH    BY COMPARING POINTERS
00AF D0 09          BNE   MORE
00B1 A5 04          LDA   EAL
00B3 C5 00          CMP   SAL
00B5 D0 03          BNE   MORE
00B7 20 08 00       JSR   JMPMON DONE.

00BA E6 02    MORE  INC   PRMLOW BUMP PROM POINTERS
00BC D0 9F          BNE   NEXT   READY IF NOT ZERO
00BE E6 03          INC   PRMHGH BUMP HIGH POINTER
00C0 D0 9B          BNE   NEXT   OKAY IF NOT ZERO
00C2 20 08 00       JSR   JMPMON EXIT ON ERROR

00C5 A9 EC    INTRPT LDAIM $EC   RESET PROGRAM LOW, VERIFY MODE
00C7 91 06          STAIY VIA
00C9 A0 0D          LDYIM IFR    SETUP TO CLEAR INTERRUPT
00CB B1 06          LDAIY VIA    READ AND WRITE TO CLEAR
00CD 91 06          STAIY VIA    INTERRUPT VIA SNEAKY TRICK
00CF 40             RTI          RETURN FROM INTERRUPT

00D0 6D       ATABLE =    $6D    AIM 65 MONITOR ENTRY
00D1 E1             =     $E1    TO DISPLAY PC COUNTER
00D2 00             =     $00    IRQ INTERRUPT VECTOR
00D3 A4             =     $A4
00D4 00             =     $00    PBDD
00D5 A0             =     $A0
00D6 02             =     $02    PBD
00D7 A0             =     $A0

00D8 05       KTABLE =    $05    KIM MONITOR ENTRY
00D9 1C             =     $1C
00DA FE             =     $FE    IRQ INTERRUPT POINTER
00DB 17             =     $17
00DC 03             =     $03    PBDD
00DD 17             =     $17
00DE 02             =     $02    PBD
00DF 17             =     $17

00E0 35       STABLE =    $35    SYM ENTRY POINT
00E1 80             =     $80
00E2 7E             =     $7E    IRQ INTERRUPT POINTER
00E3 A6             =     $A6
00E4 00             =     $00    PBDD
00E5 A0             =     $A0
00E6 02             =     $02    PBD
00E7 A0             =     $A0
```

PERFECT AIM



ATTRACTIVE FUNCTIONAL PACKAGING FOR YOUR AIM-65 MICROCOMPUTER

- Professional Appearance
- Striking Grey and Black Color Combination
- Protects Vital Components

ENGINEERED SPECIFICALLY FOR THE ROCKWELL AIM-65

- All Switches Accessible
- Integral Reset Button Actuator
- Easy Paper Tape Replacement

EASILY ASSEMBLED

- Absolutely No Alteration of AIM-65 Required
- All Fasteners Provided
- Goes Together in Minutes

MADE OF HIGH IMPACT STRENGTH THERMOFORMED PLASTIC

- Kydex 100*
- Durable
- Molded-In Color
- Non-Conductive

AVAILABLE FROM STOCK

- Allow Three to Four Weeks for Processing and Delivery
- No COD's Please
- Dealer Inquiries Invited

TO ORDER: 1. Fill in this Coupon (Print or Type Please)
2. Attach Check or Money Order and Mail to:

NAME ____________________

STREET ____________________

CITY ____________________

STATE ____________ ZIP ________

Please Ship Prepaid _____ SAE 1-1(s)
@ $43.50 each
California Residents Please Pay
$46.33 (Includes Sales Tax)

enclosures group

753 bush street
san francisco, california 94108

*TM Rohm & Haas

Patent Applied For

"THANKS FOR THE MEMORIES"
A PET MACHINE LANGUAGE MEMORY TEST

Harvey B. Herman
Chemistry Department
University of North Carolina at Greensboro
Greensboro, North Carolina 27412

Most people have surely heard the old Bob Hope theme song, "Thanks for the Memories." Whenever I hear it, I remind myself how much the explosion in personal computing is due to inexpensive memory chips. Several years ago I paid about $64.00 for a 4x16 (64 bits) static RAM by Intel. Today a 1x1024 static memory costs less than $2.00 - quite a hefty reduction in the per bit price.

That's the good news. The bad news is that all electronic parts occassionally fail and failures need to be diagnosed and repaired. The cheaper memory becomes the more we add and the harder and more time consuming it becomes to identify failed components. Diagnostic programs are one answer to this problem. Recently MICRO (7:25, Oct-Nov, 1978) published a PET memory test program written in BASIC. Execution time to test even about 200 bytes was quite long - about 1000 seconds. Clearly, a much faster test is necessary for even the smallest PET computers. If external memory is added the need for a much faster test becomes even more urgent.

An obvious way to increase the speed of a program is to write it in machine language. BASIC, a higher level of language, is notoriously slow especially when it must interpret each statement on every encounter. Writing faster machine language programs is facilitated with the help of a monitor program. PET owners have finally been given a free monitor program as part of their original purchase. This program has some nice features but the documentation is minimal. (How many times have we heard that song.) Important locations and subroutines are either not described at all or described sketchally so the program's usefulness to the average user is impaired.

However, not to worry. I have been experimenting with the monitor program by a combination of disassembly and trial and error have identified some of the missing links. You might guess from the title of this article that the purpose is to describe a fast machine language memory test. That is correct, but the other unspoken and possibly more important purpose is to teach the reader how better to use Commodore's machine language monitor program.

Table 1 summarizes important locations in Commodore's monitor. It is an expanded version of the table in their manual. For readers with access to the PET Gazette's LOMON program I have also included locations in that monitor which, incidentally, includes a disassembly in the latest version.

A large variety of machine language programs, including memory test programs, have appeared for other 6502 based systems. Jim Butterfield in "The First Book of KIM" (pp. 122-123) described a very fast machine language memory test program using a newly developed algorithm. I picked this particular KIM program for my first try at a PET translation program. Other programs developed for KIM (except when specifically hardware dependent) can be similarly translated. Our PETs will be more powerful than ever before as we can take already developed machine language software (the hard part), translate the programs for the PET and poke them into memory with the monitor (the easy part).

An inspection of the original KIM memory test program reveals some obvious PET incompatabilities. The KIM program originates at location zero and uses several KIM-specific locations (e.g., 1C4F as an exit to the KIM monitor). As a first pass we must relocate the program, change external jumps and substitute other page 0 locations. Table 2 shows the changes I made and gives some of my reasoning. Some decisions are self evident. For example, the second cassette buffer (starting at 033A) is a common place to store small PET programs as long as a second cassette is not being used. Other changes take advantage of specific features of the PET monitor. For example, the program counter (actually locations 22 and 23 as LOW and HI) is printed out after an exit to the monitor at location 0447. While the KIM monitor works similarly, the exit point and page zero locations printed are different and must be converted.

The translated program is executed using the CO command with a specified address (G 033A). After running the program several times, I became convinced it could be improved. Modifying a well documented program (as was the original) is, of course, much easier than writing one in the first place. The following changes were made:

1. Repeat the program continually until a key is pressed. Execution is very fast and one pass is not an adquate test.

2. Output an asterisk after each pass. It is nice to know the program is doing something.

3. Take the processor out of decimal mode, or hex arithmetic will not be done properly.

4. Input the beginning and ending page locations as a convenience in the GO step.

The last two modifications were easy to do before beginning execution. However, I occassionally forgot and felt it was better to insure it was done properly rather than to take a chance that the monitor had to be reloaded or BASIC restarted.

This version of the memory test program is also run with the GO command, with a specified address. The beginning and ending page location, separated by commas, are typed after the address (G 033A OA,1F). The program cycles until a faulty memory location is found which is printed as if it was the program counter or until any key is pressed. As advertised it is very fast a few second per pass for an 8 K PET (testing pages OA to 1F). A continuing outpouring of asterisks is very comforting.

My colleagues and I have found bad (or slow) memory chips with the original or modified test program on both KIM and PET computers. Happily, this does not happen very often; my hope is it won't happen to you. But if it does you will be prepared if you get this program running ahead of time. Good luck!

PET MEMORY TEST

BY HARVEY B. HERMAN
FEBRUARY 1979

```
                        ORG    $033A

                BEGIN   *      $0023
                END     *      $0024
                POINTL  *      $0019
                POINTH  *      $001A
                FLAG    *      $001B
                FLIP    *      $001C
                MOD     *      $001D
                PRINT   *      $FFD2
                GET     *      $FFE4
                INPUT   *      $FFCF
                EXIT    *      $0447
                ERROR   *      $049B
                GTBYT   *      $0656

033A D8         START   CLD
033B 20 CF FF           JSR    INPUT
033E C9 20              CMPIM  $20    SPACE CHARACTER?
0340 F0 03              BEQ    ABLE
0342 4C 9B 04           JMP    ERROR
0345 20 56 06   ABLE    JSR    GTBYT
0348 85 23              STA    BEGIN
034A 20 CF FF           JSR    INPUT
034D C9 2C              CMPIM  $2C    COMMA ?
034F F0 03              BEQ    BAKER
0351 4C 9B 04           JMP    ERROR
0354 20 56 06   BAKER   JSR    GTBYT
0357 85 24              STA    END

0359 A9 00      LOOP    LDAIM  $00
035B A8                 TAY
035C 85 19              STA    POINTL
035E 85 1B      BIGLP   STA    FLAG
0360 A2 02              LDXIM  $02
0362 86 1D              STX    MOD
0364 A5 23      PASS    LDA    BEGIN
0366 85 1A              STA    POINTH
0368 A6 24              LDX    END
036A A5 1B              LDA    FLAG
036C 49 FF              EORIM  $FF
036E 85 1C              STA    FLIP
0370 91 19      CLEAR   STAIY  POINTL
0372 C8                 INY
0373 D0 FB              BNE    CLEAR
0375 E6 1A              INC    POINTH
0377 E4 1A              CPX    POINTH
0379 B0 F5              BCS    CLEAR
037B A6 1D              LDX    MOD
037D A5 23              LDA    BEGIN
037F 85 1A              STA    POINTH
```

```
0381 A5 1B      FILL   LDA    FLAG
0383 CA         TOP    DEX
0384 10 04             BPL    SKIP
0386 A2 02             LDXIM $02
0388 91 19             STAIY POINTL
038A C8         SKIP   INY
038B D0 F6             BNE    TOP
038D E6 1A             INC    POINTH
038F A5 24             LDA    END
0391 C5 1A             CMP    POINTH
0393 B0 EC             BCS    FILL
0395 A5 23             LDA    BEGIN
0397 85 1A             STA    POINTH
0399 A6 1D             LDX    MOD
039B A5 1C      POP    LDA    FLIP
039D CA                DEX
039E 10 04             BPL    SLIP
03A0 A2 02             LDXIM $02
03A2 A5 1B             LDA    FLAG
03A4 D1 19      SLIP   CMPIY POINTL
03A6 D0 24             BNE    OUT
03A8 C8                INY
03A9 D0 F0             BNE    POP
03AB E6 1A             INC    POINTH
03AD A5 24             LDA    END
03AF C5 1A             CMP    POINTH
03B1 B0 E8             BCS    POP
03B3 C6 1D             DEC    MOD
03B5 10 AD             BPL    PASS
03B7 A5 1B             LDA    FLAG
03B9 49 FF             EORIM $FF
03BB 30 A1             BMI    BIGLP
03BD 84 19             STY    POINTL
03BF A9 2A             LDAIM $2A    ASTERISK CHARACTER *
03C1 20 D2 FF          JSR    PRINT
03C4 20 E4 FF          JSR    GET
03C7 F0 90             BEQ    LOOP
03C9 4C 47 04          JMP    EXIT

03CC 84 19      OUT    STY    POINTL
03CE 4C 47 04          JMP    EXIT
```

Program Notes

CTBYT	Change to $0658 for LOMON
033A	Clear decimal mode to insure arithmetic correct
033E	Compare with space character
033B - 0358	Input from screen: space, byte (2 characters), comma and byte. Store byte in begin and end page locations.
0359 - 03BE	Memory test program proper. Original author: Jim Butterfield.
03BF - 03CB	Print *, check for key press: no - repeat test yes - exit to monitor and print register buffer
03CC - 03D0	Abnormal exit to monitor. Program counter has address of fault.

MONITOR LOCATIONS

Table 1

Start of monitor	040F	
Exit to monitor	0447	
Break vector LOW 021B Normally 27		
Break vector HI 021C Normally 04		
Machine register storage buffer:		
Program counter LOW	0019	The registers are initialized to the value in these locations after the G command. After the break instruction (and break vector set to 0427) these locations will contain the final values of the registers.
Program counter HI	001A	
Status register	001B	
Accumulator	001C	
X-index register	001D	
Y-index register	001E	
Stack pointer	001F	
Operating System calls:		
Output byte (from A)	FFD2	
Input byte (left in A)	FFCF	(loc 260: 0 keyboard, 1 screen)
Get byte	FFE4	(A-0 no key depressed otherwide A- character)

	COMMODORE	LOMON (PET Gazette)
Output CR	04F2	04F2
Output space	063A	063B
Output byte as 2 hex	0613	0613
Input byte as 2 hex	065E	0660
ASCII to hex (from A)	0685	0687
Output? and wait for new command	049B	049B
Input 2 bytes as 4 hex (LOW in loc. 11, HI in 12)	064F	0651

KIM-PET EQUIVALENCES FOR THE MEMORY TEST PROGRAM

Table 2

	KIM	PET	NOTES
BEGIN	0000	0023	first two unused zero
END	0001	0024	page locations
POINTL	00FA	0019	printed as PC location
POINTH	00FB	001A	on exit
FLAG	0070	001B	printed as SR on exit
FLIP	0071	001C	printed as A on exit
MOD	0072	001D	printed as X on exit
EXIT	1C4F	0447	exit to monitor-print registers
START	0002	033A	start of second cassette buffer-well protected if device not used.

THE OSI FLASHER: BASIC-MACHINE CODE INTERFACING

Robert E. Jones
Handley High School
West Point St.
Roanoke, AL 36274

The following program is an example of how a machine language program for the 6502 microprocessor may be loaded from BASIC, executed, and then control may be returned to BASIC (and back again and again, as in this case.) I wrote the program to use in my job as a science teacher at Handley High School in Roanoke, Alabama, where we have two 6502 based microcomputers to use in teaching programming and solving problems of a repetitive nature in chemistry and physics. This program is set up to be run on our OHIO SCIENTIFIC CHALLENGER II.

Our CHALLENGER was originally a MODEL 65V-4K with a total of 12K of RAM. It has been updated with the new MODEL 500 CPU board with OSI MICROSOFT 8K BASIC in ROM. We also use a COMMODORE PET with 8K of RAM for programs which need graphics.

The program may be run on any OSI challenger with a video board set up to start at screen memory location 53312 (base 10) or DOOO (hex). Our video board is the old 440 BOARD with only four pages of screen memory. The new MODEL 540 video board-based Challengers may use this program to occupy all eight pages of video memory if a change is made on line 170. The number to be changed is the second number of the DATA statement, the number which tells the program how many pages of screen memory to use. For 540-based systems the new version should be as follows:

170 data 160,8,162,0,189,0,208

The reason for the ease of change is that the starting locations for the screen memory on both the 440 and 540 boards is the same, DOOO (hex). The latter version with the provision for eight pages of video display will work on either type of board, but it seems tedious to me to poke numbers into screen memory locations not visible on my 440-based machine.

```
10  FOR Y = 1 TO 32 : PRINT : NEXT Y
20  PRINT "INPUT THE DELAY CONSTANT."
30  PRINT "USE A LOW NUMBER FOR A"
40  PRINT "FAST FLASH RATE ( <.5)."
50  INPUT T
60  FOR P = 4096 TO 4130
70  READ C : POKE P,C
80  NEXT P
90  POKE 11,0 : POKE 12,16
100 FOR X = 53200 TO 54380
110 POKE X,INT(255*RND(8))
120 NEXT X
130 FOR D = 1 TO 100*T
140 NEXT D
150 X = USR (X)
160 GOTO 130
170 DATA 160,4,162,0,189,0,208
180 DATA 105,1,157,0,208,232,208
190 DATA 245,238,6,16,238,11,16
200 DATA 136,208,236,169,208,141,6
210 DATA 16,141,11,16,96,0,16
```

Figure 1

```
LINE 10         CLEARS THE SCREEN
LINES 20-50     GIVE INSTRUCTIONS AND INPUT THE DELAY FACTOR.
                THE LARGER THE DELAY FACTOR, THE SLOWER THE FLASH RATE.
LINES 60-80     READ THE MACHINE CODE PROGRAM AND STORE IT IN MEMORY LOCATIONS
                4096 TO 4130 (DECIMAL) OR 1000 TO 1022 (HEX).
LINE 90         POINTS TO THE START OF THE USR ROUTINE - WHERE TO JUMP TO WHEN
                EXITING FROM BASIC.
LINES 100-120   CREATE A SCREEN FULL OF RANDOM CHARACTERS
LINES 130-140   DELAY ROUTINE TO ALLOW THE SCREEN TO REMAIN AS IS FOR A TIME
                DEPENDING ON THE SIZE OF THE DELAY FACTOR BEFORE RETURNING TO
                THE MACHINE CODE PROGRAM.
LINE 150        CAUSES AN EXIT FROM BASIC TO THE MACHINE CODE PROGRAM
LINE 160        SENDS THE PROGRAM BACK TO THE DELAY ROUTINE WHILE IN BASIC.
LINES 170-210   DATA STATEMENTS FOR THE MACHINE CODE PROGRAM
```

Figure 2

```
                         OSI FLASHER

                         BY ROBERT E. JONES
                         FEBRUARY 1979

1000                          ORG    $1000

1000 A0 04     START   LDYIM  $04     LOAD INDEX Y WITH 4
1002 A2 00             LDXIM  $00     LOAD INDEX X WITH 0
1004 BD 00 D0  LOOP    LDAX   $D000   LOAD A WITH CONTENTS OF D000 + X
1007 69 01             ADCIM  $01     ADD 1
1009 9D 00 D0          STAX   $D000   STORE AT D000 + X
100C E8                INX            BUMP POINTER/COUNTER
100D D0 F5             BNE    LOOP    BRANCH IF NOT ZERO
100F EE 06 10          INC    $1006   INCREMENT ADDRESSES
1012 EE 0B 10          INC    $100B
1015 88                DEY            DECREMENT INDEX Y
1016 D0 EC             BNE    LOOP    LOOP IF NOT ZERO
1018 A9 D0             LDAIM  $D0
101A 8D 06 10          STA    $1006
101D 8D 0B 10          STA    $100B
1020 4C 00 10          JMP    START   CONTINUE RUNNING
```

WARNING: Set the BASIC LOMEM pointer to some address above this machine language code before running BASIC, or you will destroy the code.

6502 GRAPHICS ROUTINES

Jim Green
807 Bridge Street
Bethlehem, PA 18018

The 6502 Graphics routines were written specifically for use with the Polymorphics Video Terminal Interface. (VTI), board installed on a KIMSI, S-100 interface to a KIM. It is expected that these routines will work with other low resolution graphics boards of a similar configuration with little or no software modification but no other boards have been attempted to date.

On the VTI, 16 lines of 64 ASCII characters each, or a grid of 48 by 128 individually controllable points, can be accomodated. For each memory byte, the high bit, 7, determines how the byte is to be treated. If this bit is set the byte is displayed as an ASCII character. If the bit is clear the lowest six bits are displayed as points of a 3 by 2 point subset of the 48 by 128 point grid. Each such bit that is set will be displayed as black. The remainder will be white.

The upper left-hand corner of the display screen is the display origin. This is also the base address of the video memory. The input coordinates to the routines are specified as hex values in the X and Y registers. The X register holds the column value and has a permitted range of 0 to $7F. The Y register holds the row value from 0 to $3F.

Routines are provided: WHTPNT, to set; BLKPNT, to clear; and TSTPNT, to test the current value of a specified display bit. An additional routine, BLANKR, is used to blank the entire screen.

The principal task of the graphics routines is to gain effective access to each specified bit in the screen grid without disturbing any of the remaining points. This task is divided into two parts, first to locate the byte that contains the target bit, then to isolate the target bit itself. These tasks are performed by subroutine POINT which is called by the other routines.

Within POINT, the ranges of X and Y are first tested. If either value is found to be out of range, control is returned to the calling program with the C flag set and with no changes to the video memory. If the ranges are ok, the C flag will be clear when the routines eventually return to the calling program.

After the range tests, the row coordinate value is divided by three (by a process of successive subtraction), and the column coordinate value is divided by two. The integer quotents and remainders are saved separately in each case. The row and column quotents now point to the row and column that uniquely contain the target byte. Hence, when the row quotent is multiplied by 64 and is then added to the column quotent an offset from the video memory base address is obtained. By adding the base address to this value the absolute memory address of the target byte is obtained.

To isolate the target bit within a byte, the column and row remainders are combined to form an index value (0 to 5). This index is used to select one of six masks which may be logically combined with the byte to uniquely treat the target bit.

On the system described, these routines require an average of about a third of a milli-second to complete a single bit update. This is more than ample for most purposes.

```
;         6502 GRAPHICS ROUTINES
;         VERSION 0.3B, 18 OCT 78
;
;         COPYRIGHT BY
;         J. S. GREEN, COMPUTER SYSTEMS
;         807 BRIDGE STREET
;         BETHLEHEM, PA  18018
;         (215) 867-0924
;
;         COMMERCIAL RIGHTS RESERVED
;
;         EFFECTIVE COORDINATES IN HEX ON ENTRY:
;               COLUMN VALUE IN X, (0 - $7F)
;               ROW VALUE IN Y,    (0 - $2F)
;
;         CONSTANTS
          .DEF   VIDBAS=$C000        ;VIDEO MEMORY BASE ADDR
          .DEF   UPLIM=$C4  ;UPPER LIMIT (HIGH BYTE)
;
```

```
                   ;          VARIABLES
                              .DEF   ROW=$E2
                              .DEF   COL=$E3
                              .DEF   ROWREM=$E4
                              .DEF   COLREM=$E5
                              .DEF   GRADR=$E6
                   ;
                              .LOC   $0200
                   ;
                   ;          DISPLAY CLEAR BIT
                   ;
0200 20 4C 02      WHTPNT:  JSR    POINT        ;GET ADDRES + MASK INDEX
0203 B0 09                  BCS    WHTPT1       ;BR IF PROBLEM
0205 A0 00                  LDY#   0
0207 BD 90 02               LDAX   PLTMSK       ;GET MASK
020A 31 E6                  AND@Y  GRADR        ;AND WITH VIDEO BYTE
020C 91 E6                  STA@Y  GRADR        ;DISPLAY CLEAR BIT
020E 60            WHTPT1:  RTS
                   ;
                   ;          DISPLAY SET BIT
                   ;
020F 20 4C 02      BLKPNT:  JSR    POINT        ;GET ADDRES + MASK INDEX
0212 B0 0D                  BCS    BLKPT1       ;BR IF PROBLEM
0214 A0 00                  LDY#   0
0216 BD 90 02               LDAX   PLTMSK       ;GET MASK
0219 49 FF                  EOR#   $FF          ;REVERSE IT
021B 11 E6                  ORA@Y  GRADR        ;OR WITH VIDEO BYTE
021D 29 3F                  AND#   $3F          ;CLEAR HIGH BITS
021F 91 E6                  STA@Y  GRADR        ;DISPLAY SET BIT
0221 60            BLKPT1:  RTS
                   ;
                   ;          TEST DISPLAYED BIT
                   ;          RESULTS WITH Z FLAG SET IF BIT IS SET
                   ;
0222 20 4C 02      TSTPNT:  JSR    POINT        ;GET ADDRES + MASK INDEX
0225 B0 0B                  BCS    TSTPT1       ;BR IF PROBLEM
0227 A0 00                  LDY#   0
0229 BD 90 02               LDAX   PLTMSK       ;GET MASK
022C 49 FF                  EOR#   $FF          ;REVERSE IT
022E 29 BF                  AND#   $BF          ;CLEAR BIT 6
0230 31 E6                  AND@Y  GRADR        ;Z SET IFF GRAPHIC-BIT SET
0232 60            TSTPT1:  RTS
                   ;
                   ;          BLANK VIDEO FOR PLOT
                   ;
0233 A9 C0         BLANKR:  LDA#   >VIDBAS
0235 85 E7                  STA    GRADR+1
0237 A0 00                  LDY#   0
0239 84 E6                  STY    GRADR
023B A9 3F                  LDA#   $3F          ; 0011 1111
023D 91 E6         BLANK1:  STA@Y  GRADR
023F E6 E6                  INC    GRADR
0241 D0 FA                  BNE    BLANK1
0243 E6 E7                  INC    GRADR+1      ;HIGH ORDER ADDRES BYTE
0245 A6 E7                  LDX    GRADR+1
0247 E0 C4                  CPX#   UPLIM        ;TEST END OF SCREEN
0249 90 F2                  BCC    BLANK1       ;BR NOT DONE
024B 60                     RTS
                   ;
```

```
                    ;          GET BYTE ADDRES & BIT MASK
                    ;
024C E0 80          POINT:    CPX#   $80          ;128 IS TOO HIGH
024E B0 3F                    BCS    POINT3       ;BR TOO HIGH
0250 C0 30                    CPY#   $30          ;48 IS TOO HIGH FOR ROW
0252 B0 3B                    BCS    POINT3       ;BR TOO HIGH
0254 8A                       TXA                 ;COLUMN
0255 48                       PHA                 ;SAVE IT
0256 98                       TYA                 ;ROW
0257 AA                       TAX                 ;DIVIDE ROW BY 3
0258 A0 FF                    LDY#   $FF          ;INITIALIZE QUOTENT
025A C8             POINT1:   INY                 ;ACCUMULATE QUOTENT
025B CA                       DEX                 ;SUBTRACT 3
025C CA                       DEX
025D CA                       DEX
025E 10 FA                    BPL    POINT1       ;BR MORE
0260 E8                       INX                 ;RESTOR 3
0261 E8                       INX
0262 E8                       INX
0263 86 E4                    STX    ROWREM       ;ROW REMAINDER
0265 84 E2                    STY    ROW          ;INTEGER QUOTENT
0267 A2 00                    LDX#   0
0269 86 E5                    STX    COLREM       ;INITIALLY CLEAR
026B 68                       PLA                 ;RESTOR COLUMN
026C 4A                       LSRA                ;DIVIDE BY 2
026D 85 E3                    STA    COL          ;INTEGER QUOTENT
026F 26 E5                    ROL    COLREM       ;REMAINDER FROM CARRY
0271 A5 E2                    LDA    ROW
0273 18                       CLC
0274 86 E7                    STX    GRADR+1      ;CLEAR ADDRES HI
0276 A2 05                    LDX#   5            ;PREP TO MPY BY 2**6 (=64)
0278 0A             POINT2:   ASLA                ;MPY BY 2 EACH LOOP
0279 26 E7                    ROL    GRADR+1      ;OVERFLO TO ADDRES HI
027B CA                       DEX
027C 10 FA                    BPL    POINT2       ;BR TIL DONE 6 TIMES
027E 65 E3                    ADC    COL          ;ADD THE PLACE IN THE ROW
0280 85 E6                    STA    GRADR
0282 A9 C0                    LDA#   >VIDBAS      ;VIDEO MEMORY BASE ADDR HI
0284 65 E7                    ADC    GRADR+1
0286 85 E7                    STA    GRADR+1      ;ADDRES POINTS TO BYTE
                                                  ; IN VIDEO MEMORY
                    ;
                    ;         NOW CALC MASK INDEX FOR BIT WITHIN BYTE
0288 A5 E4                    LDA    ROWREM       ;EITHER 0, 1 OR 2
028A 66 E5                    ROR    COLREM       ;EITHER 0 OR 1 INTO CARRY
028C 2A                       ROLA                ;COMBINE WITH CARRY
028D AA                       TAX
028E 18                       CLC                 ;CLEAR CARRY SAYS ANS OK
028F 60             POINT3:   RTS
                    ;
0290 1F             PLTMSK:   .BYTE $1F           ;UP-LEFT POINT WITHIN BYTE
0291 3B                       .BYTE $3B           ;UP-RT
0292 2F                       .BYTE $2F           ;MID-LF
0293 3D                       .BYTE $3D           ;MID-RT
0294 37                       .BYTE $37           ;LO-LF
0295 3E                       .BYTE $3E           ;LO-RT
                    ;
                              .END                NO ERRORS DETECTED
                                                  PASS (1-2)?
```

Interactive Baseball

SYSTEM: Standard Apple II
MEMORY SIZE: 16K or More
LANGUAGE: Interger Basic
DESCRIPTION: An Interactive Baseball Game that uses Color Graphics extensively. Play a 7 or 9 inning game alone or against a friend, (it will handle extra innings). Has sound effects with men running bases. Base stealing and pitching are under player control. Double plays and picking off of base runners under software control. Keeps track of team runs, innings, balls and strikes, outs, hits, has strike-outs and walks, and uses paddle inputs to interact with the program.

PRICE: Cassette $12.50, Basic Listing $6.00.

INCLUDES: User manual with complete documentation. Plus a listing of key line numbers with an explanation of their purpose within the program.

Available From: PAT CHIRICHELLA
506 Fairview Ave.
Ridgewood, N.Y. 11237
(Dealer Inquires Invited)

6502 BIBLIOGRAPHY
PART IX

William R. Dial
438 Roslyn Avenue
Akron, OH 44320

421. PURSER, ROBERT E. "Reference List of TRS-80, PET and APPLE II Computer Cassettes"
Edition 4, November, 1978 (P.O. Box 466, El Dorado, CA 95623
A very complete listing of software for the Apple II and PET is given. A few software reviews are given.

422. MICRO No 8 (Dec., 1978-Jan., 1979)
De Jong, Marvin L. "6502 Interfacing for Beginners: Buffering the Busses"
The author continues his series of tutorial articles discussing the need for buffers, types of buffer chips and some ex-experiments and an application.
Anon. "Microbes"
An entire section of code from "Breaker: An Apple II Debugging Aid" MICRO NO 7 pg 5 was omitted and is given in this correction. Also a correction for Husband's "Design of a PET TTY Interface" Micro No. 6 pg 5.
Suitor, Richard F. "Life for your Apple"
A new version of LIFE has the generation calculations in assembly language to speed the program
Reich, Dr. L.S. "Computer-Determined Kinetic Parameters in Thermal Analysis"
A program for the quantitative estimation of kinetic parameters for the material being degraded such as activation energy and reaction order. Uses Apple II.
Christensen, Alan K. "Continuous Motion Graphics or How to Fake a Joystick with the PET"
Basic supported routines are too slow to allow smooth movement. Action is enhanced by direct access of screen and keyboard.
Powlette, Joseph L. and Jeffery, Donald C. "Storage Scope Revisited"
With the hardware changes suggested in this article the performance of DeJong's program to transform an ordinary oscilloscope to a storage scope gives results approaching those of a commercial unit.
Auricchio, Rick, "An Apple II Program Relocator"
A program to move an Assembly language program to another part of memory. Changes all absolute references within the program.
Gieryic, John, "SYM-1 Tape Directory"
Program to allow the SYM owner to examine his cassette tape to find what information is there.
Anon. "The Best of MICRO Volume 1"
A book containing most of the articles that were published in MICRO Volume 1.
Butterfield, Jim "Inside PET Basic"
Two new programs for PET. FIND will search a PET BASIC program for a particular data string that will list the lines containing the string. RESEQUENCE will renumber your program fixing up GOTO's and other functions.
Connolly, M.R. Jr. "An Apple II Page 1 Map"
This article shows a clever method of creating all sorts of nifty effects, title pages, etc., on your Apple.
Dial, Wm R. "6502 Bibliography, Part VII"
Some 88 more references to the growing 6502 literature.

423. PET Gazette (Oct./Nov., 1978)
Anon. "Software Reviews — PET"
Many reviews of PET software are to be found scattered through this issue of the Gazette. Also review of many new hardware items for PET.
Staebel, Jon "PET Hints"
PET Gazette 1 No 6 pg 6, (Oct./Nov., 1978)
A discussion of the timer and built-in clock in the PET, examples of use. How to stimulate a repeat key on the PET.
Barsanian A. "Tape Tips"
PET Gazette 1 No 6 pg 8-9 (Oct./Nov., 1978)
Some sensible tips on using PET tape cassettes, storage, copying. How to locate one program out of many on a tape.
Cumberton, Dennis "Tape Tips"
PET Gazette 1 No 6 pg 9 (Oct./Nov., 1978)
Recommendations on the use of C-30 cassettes stripped down to C-10 equivalent. Comments on brands found satisfactory.
Stone, Mike "Program Overlays"
PET Gazette 1 No 6 pg 11-13 (OCT./Nov., 1978)
Joining two programs.
Anon. "New PET Booklet-PET Communicates with the Outside World"
PET Gazette 1 No 6 pg 15-19 (Oct./Nov., 1978)
Summary of the important information released including pinout for Parallel User Port, Second Cassette Interface, and Memory Expansion Connector. IEEE Bus Limitations, I/O Commands, I/O operations, Recording techniques, Error Detection, etc., etc.
Lindsay, Len "Kilobaud Column for PET Users"
Hints on programming with your PET, Use of the GET command.

424. **Dr. Dobb's Journal 3 Issue 10 No. 30 (Nov./Dec., 1978)**

Bridge, Theodore E. "A Curve-Fitting Program Using a Focal Interpreter on the KIM-1"

Focal is used with the KIM in a curve-fitting program.

Swanks, Joel "Tiny GRAFIX for Tiny BASIC"

Grafix is a system for graphic display on a small computer system, including Pittman's Tiny Basic, a SWTPC GT-6144 TV Graphics board, some machine language subroutines and a KIM-1 with 4K of memory.

Oliver, John P. "Astronomy Application for PET FORTH"

Using a newly available language PET-FORTH version 1.0, a PET was used to provide control functions for a telescope.

425. **Kilobaud No. 25 (Jan., 1979)**

Lindsay, len "PET Pourri"

A new column on the PET has sections discussing Accessories, Publications, Software, Programming Hints, and PET Problems. A very helpful series of hints on Cassette recorder maintenance and saving data is included.

Brisson, Dennis "New Products"

6502 products include reviews on weight control/biorythm programs, a telephone cost-control center, the RS-16-HP "universal" interface for PET, a 6502 Assember for PET, a PET Word Processor, etc.

Fuller, Steve "OSI User Group"

The Newton Software Exchange, PO Box 518, Newton Center, MA 02158, is forming a user's group for OSI products, especially the Challenger series.

Anon, "Letters"

This month 6502 letters refer to the November article "Do It with a KIMSI", the September Article "Super Cheap 2708 programmer, etc

Lang, George E. "u-Panel"

See the reaction of every register of your microprocessor as you single step your KIM through a program.

Ketchum, Don "Display Your PET"

Watch the Monitor screen as all 316 PET characters appear on the Screen

Carpenter, Charles R. "SHHH... People are Sleeping"

The Telpar PS-40-3C-1 serves as a quiet and economical substitute for a noisy and expensive teletype.

Yob, Gregory "PET Techniques Explained"

Supplementing information from Commodore, this article gives information on cassette files.

426. **Calculators/Computers Magazine 2 Issue 7 (Nov./Dec., 1978)**

Costello, Scott H. "Hilo–A Number-Guessing Program that Illustrates Several Math Concepts"

Modifications for the program to run on several different computers, including PET are given. A number of variations are suggested.

Albrecht, Bob and Albrecht, Karl "PET BASIC for Parents and Teachers"

An explanation of many of the keys on the PET keyboard.

427. **Dr. Dobb's Journal 4 Issue 1 Number 31 (Jan., 1979)**

Seiler, Bill "PET BASIC Renumber"

A program to put your line numbers in a more ordinary list.

Moser, Carl W. "Add a Trap Vector for Unimplemented 6502 Opcodes"

Ideas on how to provide hardware and a program to ferret out those hidden opcodes

Aresco, P.O. Box 43, Audubon, PA 19407 "6K Assembler/Text Editor for Apple II"

A 6K machine language for the Apple II.

Terc Services, 575 Technology Sq., Cambridge MA 02139 "KIM-1 Interface Set"

Permits easy access to the I/O ports on the KIM.

428. **Byte 4 No. 1 (Jan., 1979)**

Helmers, Carl "Pascal Progress"

The University of Calif at San Diego plans to make the UCSD Pascal system available on Apple II computer early in 1979.

PRS The Program of the MonthCorporation, 257 Central Park West New York, N.Y. 10024

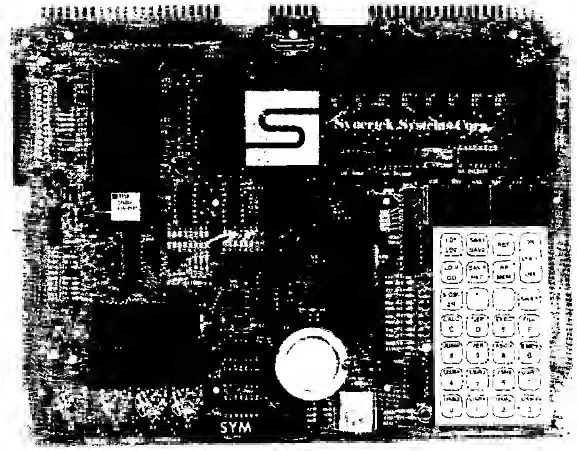
A2FP is a Plotting Program for Apple II which plots 2-dimensional functions in high resolution graphics.

Leff, Alan A. and Boos, D L. "A Timely Modification to KIMER"

Modification of the Baker program "Kimer: A KIM-1 Timer" Byte, July 1978, pg 12 to allow it to run as 12 hour clock.

SYM-1, 6502-BASED MICROCOMPUTER

- FULLY-ASSEMBLED AND COMPLETELY INTEGRATED SYSTEM that's ready-to-use
- ALL LSI IC'S ARE IN SOCKETS
- 28 DOUBLE-FUNCTION KEYPAD INCLUDING UP TO 24 "SPECIAL" FUNCTIONS
  EASY-TO-VIEW 6-DIGIT HEX LED DISPLAY
- KIM-1* HARDWARE COMPATIBILITY
  The powerful 6502 8-Bit MICROPROCESSOR whose advanced architectural features have made it one of the largest selling "micros" on the market today.
- THREE ON-BOARD PROGRAMMABLE INTERVAL TIMERS available to the user, expandable to five on-board.
- 4K BYTE ROM RESIDENT MONITOR and Operating Programs.
- Single 5 Volt power supply is all that is required.
- 1K BYTES OF 2114 STATIC RAM onboard with sockets provided for immediate expansion to 4K bytes onboard, with total memory expansion to 65, 536 bytes.
- USER PROM/ROM: The system is equipped with 3 PROM/ROM expansion sockets for 2316/2332 ROMs or 2716 EPROMs
- ENHANCED SOFTWARE with simplified user interface
- STANDARD INTERFACES INCLUDE:
  —Audio Cassette Recorder Interface with Remote Control (Two modes: 135 Baud KIM-1* compatible, Hi-Speed 1500 Baud)
  —Full duplex 20mA Teletype Interface
  —System Expansion Bus Interface
  —TV Controller Board Interface
  —CRT Compatible Interface (RS-232)
- APPLICATION PORT: 15 Bi-directional TTL Lines for user applications with expansion capability for added lines
- EXPANSION PORT FOR ADD-ON MODULES (51 I/O Lines included in the basic system)
- SEPARATE POWER SUPPLY connector for easy disconnect of the d-c power
- AUDIBLE RESPONSE KEYPAD



Synertek has enhanced KIM-1* software as well as the hardware. The software has simplified the user interface. The basic SYM-1 system is programmed in machine language. Monitor status is easily accessible, and the monitor gives the keypad user the same full functional capability of the TTY user. The SYM-1 has everything the KIM-1* has to offer, plus so much more that we cannot begin to tell you here. So, if you want to know more, the SYM-1 User Manual is available, separately.

SYM-1 Complete w/manuals	**$269.00**
SYM-1 User Manual Only	**7.00**
SYM-1 Expansion Kit	**75.00**

Expansion includes 3K of 2114 RAM chips and 1-6522 I/O chip.

SYM-1 Manuals: The well organized documentation package is complete and easy-to-understand.

SYM-1 CAN GROW AS YOU GROW. Its the system to BUILD-ON. Expansion features that are soon to be offered:

***BAS-1 8K Basic ROM (Microsoft)**	**$159.00**
***KTM-2 TV Interface Board**	**349.00**

**We do honor Synertek discount coupons*

QUALITY EXPANSION BOARDS DESIGNED SPECIFICALLY FOR KIM-1, SYM-1 & AIM 65

These boards are set up for use with a regulated power supply such as the one below, but, provisions have been made so that you can add onboard regulators for use with an unregulated power supply. But, because of unreliability, we do not recommend the use of onboard regulators. All I.C.'s are socketed for ease of maintenance. *All boards carry full 90-day warranty.*

All products that we manufacture are designed to meet or exceed industrial standards. All components are first quality and meet full manufacturer's specifications. All this and an extended burn-in is done to reduce the normal percentage of field failures by up to 75%. To you, this means the chance of inconvenience and lost time due to a failure is very rare; but, if it should happen, we guarantee a turn-around time of less than forty-eight hours for repair.

Our money back guarantee: If, for any reason you wish to return any board that you have purchased directly from us within ten (10) days after receipt, complete, in original condition, and in original shipping carton; we will give you a complete credit or refund less a $10.00 restocking charge per board.

VAK-1 8-SLOT MOTHERBOARD

This motherboard uses the KIM-4* bus structure. It provides eight (8) expansion board sockets with rigid card cage. Separate jacks for audio cassette, TTY and power supply are provided. Fully buffered bus.

VAK-1 Motherboard **$129.00**

VAK-2/4 16K STATIC RAM BOARD

This board using 2114 RAMs is configured in two (2) separately addressable 8K blocks with individual write-protect switches.

VAK-2 16K RAM Board with only 8K of RAM (½ populated)	**$239.00**
VAK-3 Complete set of chips to expand above board to 16K	**$175.00**
VAK-4 Fully populated 16K RAM	**$379.00**

VAK-5 2708 EPROM PROGRAMMER

This board requires a +5 VDC and ±12 VDC, but has a DC to DC multiplyer so there is no need for an additional power supply. All software is resident in on-board ROM, and has a zero-insertion socket.

VAK-5 2708 EPROM Programmer **$269.00**

VAK-6 EPROM BOARD

This board will hold 8K of 2708 or 2758, or 16K of 2716 or 2516 EPROMs. *EPROMs not included.*

VAK-6 EPROM Board **$129.00**

VAK-7 COMPLETE FLOPPY-DISK SYSTEM (May '79)

VAK-8 PROTYPING BOARD

This board allows you to create your own interfaces to plug into the motherboord. Etched circuitry is provided for regulators, address and data bus drivers; with a large area for either wire-wrapped or soldered IC circuitry.

VAK-8 Protyping Board **$49.00**

POWER SUPPLIES

ALL POWER SUPPLIES are totally enclosed with grounded enclosures for safety, AC power cord, and carry a full 2-year warranty.

FULL SYSTEM POWER SUPPLY

This power supply will handle a microcomputer and up to 65K of our VAK-4 RAM. ADDITIONAL FEATURES ARE: Over voltage Protection on 5 volts, fused, AC on/off switch. Equivalent to units selling for $225.00 or more.

Provides +5 VDC @ 10 Amps & ±12 VDC @ 1 Amp

VAK-EPS Power Supply **$125.00**

KIM-1* Custom P.S. provides 5 VDC @ 1.2 Amps and +12 VDC @ .1 Amps

KCP-1 Power Supply **$41.50**

SYM-1 Custom P.S. provides 5 VDC @ 1.4 Amps

VCP-1 Power Supply **$41.50**

*KIM is a product of MOS Technology

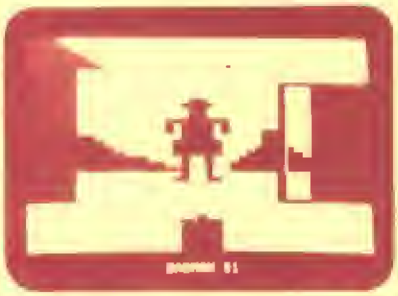
RNB ENTERPRISES
INCORPORATED

2967 W. Fairmount Avenue
Phoenix AZ. 85017
(602)265-7564

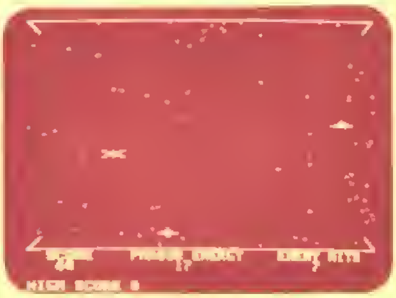
master charge VISA

QUALITY SOFTWARE

APPLE II®



GUNFIGHT $9.95



PHAZOR ZAP $15.95

FLYSWATTER $9.95

DEPTH CHARGE $15.95

PIRATES $9.95

Apple FORTH
Pet FORTH

FORTH is a unique threaded language that is ideally suited for systems and applications programming on a micro-processor system. The user may have the interactive FORTH Compiler/Interpreter system running stand-alone in 4K to 6K bytes of RAM. The system also offers a built-in incremental assembler and text editor. Since the FORTH language is vocabulary based, the user may tailor the system to resemble the needs and structure of any specific application. Programming in FORTH consists of defining new words, which draw upon the existing vocabulary, and which in turn may be used to define even more complex applications. Reverse Polish Notation and LIFO stacks are used in the FORTH system to process arithmetic expressions. Programs written in FORTH are compact and very fast.

APPLE II COMPUTER $34.95
PET 2001 COMPUTER $34.95

Apple PIE

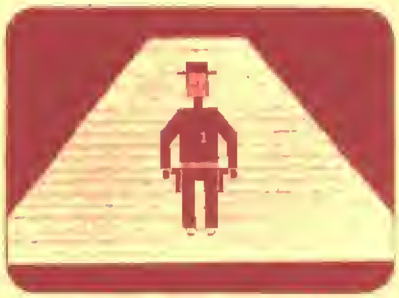
PIE (PROGRAMMED IMPROVED EDITOR) is an enhanced curser-based editor that works unlike most currently available text editors. All PIE commands consist of control characters, which are assigned to user defined function locations. The keys of the system input keyboard, are assigned specific PIE Editor function commands by the user. Commands in the PIE Editor may optionally be preceded by an Escape character, followed by a numeric or string argument.

APPLE II COMPUTER $19.95

All orders include 3% postage and handling.
Apple II is a registered trademark of Apple Computer, Inc. Pet is a registered trademark of Commodore International.
California residents add 6% sales tax.
VISA & MASTERCHARGE Accepted.

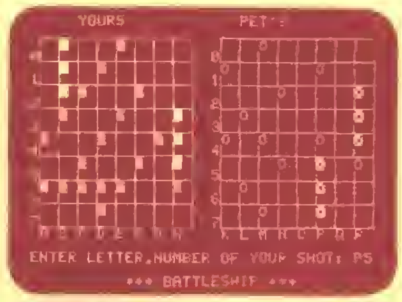
APPLE II LIGHT PEN $34.95

PET®

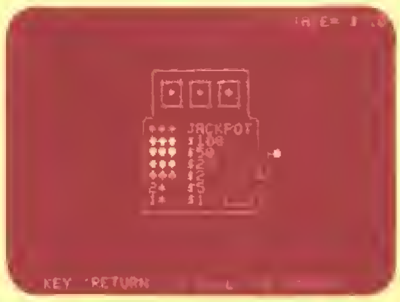


QUICK-DRAW $6.95

DOMINOES $6.95



BATTLESHIP $6.95



SLOT MACHINE $6.95

PROGRAMMA INTERNATIONAL, Inc.
3400 Wilshire Blvd.
Los Angeles, CA 90010
(213) 384-0579

Dealer Inquiries Invited

PROGRAMMA Software Products